How to deploy robotic mobile fulfillment systems

Lu Zhen [1], Zheyi Tan [1], René de Koster [2], Shuaian Wang [3*]

[1] *School of Management, Shanghai University, Shanghai, China*
[2] *Rotterdam School of Management, Erasmus University, The Netherlands*
[3] *Faculty of Business, The Hong Kong Polytechnic University, Kowloon, Hong Kong*
* Corresponding author.
Contact: lzhen@shu.edu.cn (LZ), zytan_shu@163.com (ZT), rkoster@rsm.nl (RK), wangshuaian@gmail.com (SW).

Abstract: Many warehouses involved in e-commerce order fulfillment use robotic mobile fulfillment systems. As demand and variability can be high, scheduling orders, robots, and storage pods in interaction with manual workstations is critical to obtaining high performance. Simultaneously, the scheduling problem is extremely complicated due to interactions between decisions, many of which must be taken timely due to short planning horizons and a constantly changing environment. This paper models all such scheduling decisions in combination to minimize order fulfillment time. We propose two decision methods for the above scheduling problem. The models batch the orders using different batching methods, assign orders and batches to pods and workstations in sequence, and robots to jobs. Order picking and stock replenishment operations are included in the models. We conduct numerical experiments based on a real-world case to validate the efficacy and efficiency of the model and algorithm. Instances with 14 workstations, 400 orders, 300 stock-keeping units (SKUs), 160 pods, and 160 robots can be solved to near optimality within four minutes. Our methods can be applied to large instances, e.g., using a rolling horizon. Since our model can be solved relatively fast, it can be used to take managerial decisions and obtain executive insights. Our results show that making integrated decisions, even when done heuristically, is more beneficial than sequential, isolated optimization. We also find that positioning pick stations close together along one of the system's long sides is efficient. The replenishment stations can be grouped along another side. Another finding is that SKU diversity per pod and SKU dispersion over pods have a strong and positive impact on the total completion time of handling order batches.

Keywords: Robotic warehouse systems; intralogistics optimization; e-commerce order fulfillment; order picking and replenishment.

1. Introduction

As the market share of e-commerce grows, online retailers face the challenge of handling an increasingly large number of time-critical orders. On November 11, 2020 (the annual "Double 11" global shopping festival organized by the Alibaba Group), the revenue from online shopping reached almost 74.1 billion USD, and the number of delivery orders shipped by the CAINIAO™ Network (a logistics service provider for the Alibaba Group) totaled about 2.32 billion (Businesswire, 2020). The ongoing COVID-19 pandemic also led to a significant increase in global parcel delivery volume, which reached 130 billion units in 2021. In addition, delivery time requirements are stringent. Many e-commerce retailers promise next-day delivery if customers order before midnight (Hou et al., 2018). Processing such large numbers of orders with very short delivery windows is a challenge. E-commerce warehouses increasingly use automated and robotic handling systems to increase the operational efficiency of order picking, which enhances the responsiveness of a warehouse (Zhen and Li, 2022). Since the first implementations by Amazon in 2014, robotic mobile fulfillment systems (RMFSs)

have been widely adopted in automated warehouses in various industry sectors, such as footwear, clothing, cosmetics, and pharmaceuticals by third-party logistics providers in manufacturing and e-commerce warehouses. Amazon implemented Kiva™ RMFSs, now renamed Amazon Robotics™. Since then, many other companies have developed RMFSs, such as CarryPick™ by Swisslog, Butler™ by GreyOrange, the Scallog System™, Racrew™ by Hitachi, Geek+™ by JizhijiaTech, Quicktron™ by Flashhold, Malu™ by MaluInnovation, and Qianmo™ by Hikrobot. RMFSs use autonomous mobile robots to lift movable racks (also called 'pods') and bring them directly to stationary pickers working in pick stations. Working in a "parts-to-pickers" mode, rather than pickers traveling around a warehouse, significantly increases pickers' productivity (Wurman et al., 2008; Qin et al., 2022). Parts-to-picker operations with moveable pods are particularly suited to contexts of large assortments of small products and small orders, each consisting of just a few lines with low order quantities (Lamballais et al., 2017; Boysen et al., 2019b). RMFSs are also scalable and flexible as more robots and pods can be added to increase throughput capacity. This is particularly important for e-commerce retailers that face volatile demand with large fluctuations (Azadeh et al., 2019). Figure 1 shows an RMFS warehouse of CAINIAO™ in Huizhou City, China.

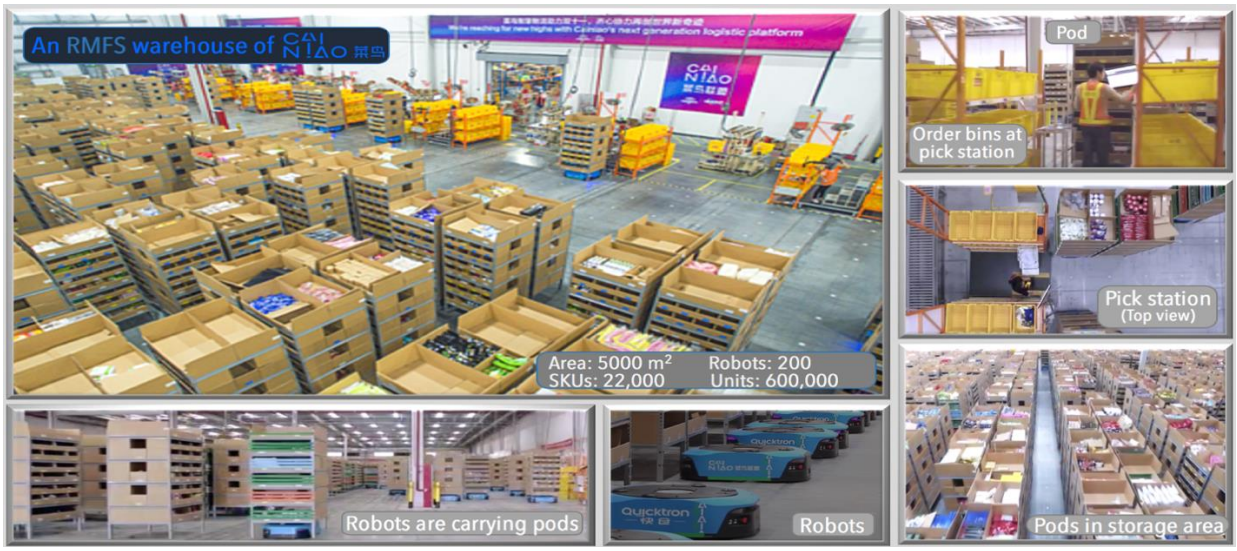


Figure 1: A CAINIAO™ warehouse in Huizhou City, China with a Quicktron™ RMFS (Source: Quicktron)

An RMFS consists of three main parts: (1) hardware such as robots, pods, and workstations; (2) a control system that gives robots autonomy, such as avoiding collisions, auto battery-charging, route scheduling, and speed control to avoid traffic jams; and (3) process management software, controlling the interaction of orders, workstations, robots, and pods, based on data captured and exchanged by an Internet-of-Things (IoT) based data exchange network. Although the hardware and control system technologies are relatively mature, the system has complex interactions between order structures (size and composition), complicated product-to-pod assignment decisions, product diversity per pod and dispersion over pods decisions, and robot-to-pod and order-to-workstation assignments. These can be solved by simple heuristics based on intuitive decision rules designed by human experience. However, performance can be improved by using advanced analytical methods (Shi et al., 2021). IoT-enabled RMFS warehouses can collect data on the status of the stock-keeping units (SKUs) in each pod, of each robot (e.g., empty, laden, battery level), of each station (orders being processed, pods on-site and on the way), and of information on SKUs on orders, and ordered quantities. These analytical

methods optimize time-sensitive decisions on assigning orders to stations, pods to stations, robots to pods, and dynamic decisions on robot and pod travel routes. However, despite these possibilities, optimizing operational level decisions in RMFSs has received little attention in literature.

Initial simulation-based results by Merschformann et al. (2019) have shown that smart scheduling of robots, orders, and workstations can substantially improve performance. This paper uses mathematical programming to generate operational decisions to minimize order makespan. More specifically, we investigate the assignment of orders to stations, pods to workstations, robots to pods, and the travel routes of robots with pods (i.e., the sequence in robots carrying a pod should visit stations). We first design an intuitive, fast, three-stage decision method, inspired by how decisions are made in practice, and then develop a mixed-integer programming (MIP) integrated decision model and a fast solution algorithm. The algorithm can solve instances with 14 stations, 400 orders, 300 SKUs, 160 pods, and 160 robots within four minutes and allows us to apply our methodology for large RMFSs, e.g., by a rolling horizon approach. We also extend the basic model with a single batch mode (i.e., a given set of orders must be scheduled) to a more generic context and formulate decision models for semi-batch contexts, in which some orders are already being processed at stations and some robots are carrying pods and have arrived at a station or are on their way to a station. This situation allows timely decision-making based on instantaneous information captured by IoT devices (Shi et al., 2021). In addition, we extend the basic batch model to a model variant with replenishment stations applying different replenishment strategies.

The remainder of this paper is organized as follows: Section 2 reviews related work and Section 3 introduces the background to the problem. Section 4 proposes two decision methods for order fulfillment in RFMSs. In Section 5, we present numerical experiments to validate the relative merits of the two methods and investigate the managerial implications. Section 6 discusses extensions to this study, and we conclude in the last section.

2. Related work

RMFSs have attracted much attention from academia, and some excellent literature reviews were recently conducted on robotic warehouse systems and RMFSs (Azadeh et al., 2019; Boysen et al., 2019a and 2019b).

Most RMFS literature models tactical design-related objectives using queuing networks. Nigam et al. (2014) use a closed queuing network (CQN) formulation to analyze robot utilization, system throughput, and the expected throughput time for order-picking of an RMFS in a scenario of single-line orders and a turnover class-based storage policy. Job throughput times can also be studied using open or semi-open queuing network models. Lamballais et al. (2017) use a semi-open queuing network (SOQN) model, extending the work of Nigam et al. to consider multi-line orders and storage zones. Their models can be applied to assess RMFS performance (e.g., throughput capacity and order throughput time) under different warehouse layouts or robot zoning policies. Lamballais et al. (2020) further extend their previous work to consider pod replenishment and introduce a new modeling method, a cross-class matching multi-class SOQN model. They then investigate three important decisions for RMFSs: the most suitable way of spreading inventory units across multiple pods (i.e., product dispersion), the most suitable ratio of pick stations to replenishment stations, and the most suitable replenishment level for pods. Lamballais et al. (2022) use a SOQN to study the dynamic reassignment of robots to stations and dynamic switching of stations between picking and replenishment jobs for time-varying demand

environments. Roy et al. (2019) conduct a comprehensive RMFS performance evaluation for a single-zone case and a multi-zone case based on a CQN and a two-stage stochastic model, respectively, and use their models to compare dedicated and pooled robot policies for the single zone case and random robot assignment and congestion-avoidance robot assignment policies for the multi-zone case. Yuan and Gong (2017) build an open queuing network (OQN) to estimate RMFS throughput time and analyze dedicated versus pooled robot policies. Their method can determine the optimal number and speed of robots. Zou et al. (2017) develop an SOQN model to investigate an assignment rule of robots to pick stations in an RFMS. The rule is based on handling speed. They use the matrix-geometric method to solve the model. They also use their model to decide on the optimal dimension of shelf blocks and the length/width ratio of the storage area. Zou et al. (2018) build an SOQN model to compare two battery recovery strategies (battery charging or swapping) in an RMFS operation and design a decomposition method to solve the model. They also investigate the relationship between battery recovery strategies, order throughput times, and resource costs.

Our model focuses on pod, order, and workstation scheduling decisions. Such decisions are commonly modeled using MIP models. Boysen et al. (2017) build a MIP model to solve an operational level decision problem that synchronizes order picking and shelf rack moving in an RMFS context with one isolated pick station. Their study is similar to the problem background and methodology in this study. However, our study differs in several ways. We consider the assignment of robots to pods and multiple workstations, in which a pod (rack) can visit several stations in one trip. The aim is to complete all orders at all stations as quickly as possible. In contrast, Boysen et al.'s study (2017) aims to minimize the number of rack visits. We also formulate models based on a continuous time frame rather than using discrete-time slots and consider the quantity of each SKU in an order. Weidinger et al. (2018) also develop a MIP model for assigning racks (pods) to storage locations based on the completion time of pods at stations and the time required at stations. They propose a novel adaptive programming algorithm to solve their model efficiently. Our model differs because the known data in their problem are the decision variables in our study. Jiang et al. (2020) propose a MIP model for picking and replenishment synchronization for RMFSs. They argue that their synchronization approach might be better than the conventional independent schemes in certain contexts. Valle and Beasley (2021) suggest a MIP model and two heuristics for simultaneously assigning orders and pods to pick stations. Yang et al. (2021) formulate a MIP model and a two-stage heuristic for integrated optimization of order sequencing and rack scheduling in RMFSs. Xie et al. (2021) consider the order splitting in a MIP model on order assignment and pod selection in RMFSs. Wang, Yang, Qi (2022) design an integer programming model and a two-stage hybrid heuristic for order and pod sequencing in RMFSs. Wang, Sheu, Teo, Xue (2022) study a problem of finding a suitable robot schedule that takes into account the schedule-induced fluctuation of the working states of human pickers; and they design an approximate dynamic programming-based branch-and-price (ADP-B&P) algorithm to solve their problem. Based on the MIP methodology, Boysen et al. (2023) propose a novel two-step multiple-scenario approach to optimize the robotized sorting systems in a real-time context. Our study differs significantly from the above studies (see Table 1) as it considers more comprehensive decisions and solves much larger-scale problem instances. Most studies aim to minimize the number of pod visits. However, in practice the main objective is to minimize the makespan (i.e., the completion time of all orders). Our model

is formulated based on a continuous time frame, which is more challenging than using discrete time slots. It also considers features such as replenishment and the detailed calculation of picking time. In sum, this study makes a more comprehensive contribution to the literature on optimization models in RMFSs.

Table 1: Comparison between this paper and studies on pod assignment and scheduling problems in RMFSs

Papers	**Objective** (minimize)	**Assignment decision**			**Sequencing decision**		**Time frame**	**Other features**	**Solution Method**	**Problem size**
		orders to station	*pods to station*	*robot to pod*	*stations for each pod*	*pods for each station*				
Boysen et al. (2017)	pod visits	—	√	—	—	√	discrete		heuristic	1 station 100 orders 100 pods
Jiang et al. (2020)	pod visits	—	√	—	—	—	N/A	pod replenishment	heuristic	2 stations 250 orders
Yang et al. (2021)	pod visits	—	√	—	—	√	discrete		heuristic	1 station 500 orders
Valle and Beasley (2021)	pod visits	√	√	—	—	√	N/A	pod revisits	heuristic	10 stations 200 orders 500 pods
Xie et al. (2021)	pod visits	√	√	—	—	—	N/A	order split	heuristic	4 stations 250 orders 100 pods
Wang, Yang, Qi (2022)	pod visits	√	√	—	√	—	discrete		heuristic	8 stations 100 orders 100 pods
Wang, Sheu, Teo, Xue (2022)	makespan	—	√	—	—	√	N/A	picker states	ADP-B&P	9 stations 95 orders 120 pods
Our paper	makespan	√	√	√	√	√	continuous	pod replenishment detailed pick time	heuristic	14 stations 400 orders 160 pods

Besides analytical studies using queuing or MIP models, scholars have used other methodologies to investigate the performance of various operational decision rules and storage policies. Yuan et al. (2019) examine the decision on pods' returning locations in RMFSs and develop a fluid model to compare the performance of the velocity-based storage policy with a random storage policy. Merschformann et al. (2019) develop a highly realistic discrete event simulation model, which keeps track of all robot and pod movements and all inventory on each pod to analyze seven performance measures (e.g., system throughput capacity and order due time). The model can capture more details than analytical methods. However, creating, adapting, and running the model to evaluate different scenarios is time-consuming, making it less fit for timely decisions.

Assuming it solves sufficiently fast, a MIP model may be appropriate to study operational performance objectives, particularly the makespan estimation of batch orders. Our study proposes a comprehensive decision-making model, which includes multiple interrelated decisions with different relative merits that influence performance. We consider realistic factors, such as the size of SKUs, pick and replenishment stations, order batches, and detailed pick times when allocating robots to pods and pod routes. To the best of our knowledge, no studies have proposed models similar to the submodels in this study or have studied an integrated model. Moreover, we design an algorithm that can quickly solve the models for real-world scale instances to allow rapid operational decision-making. The algorithm can solve instances with 14 stations, 400 orders, 300 SKUs, 160 pods, and 160 robots within four minutes. The optimality gap is close to zero for small-scale instances. We also obtain managerial insights using this fast algorithm in extensive experiments.

3. Problem background

Figure 2 shows a top-down view of a warehouse layout using an RFMS. Products (or SKUs) are stored in movable pods. Autonomous mobile robots (AMRs) transport the pods from their storage locations to the pick stations where workers pick products to fulfill orders or to the replenishment stations to replenish products in the pods. The order picker picks products from a current pod located on one side of the picker and puts them into the order bins on the other side. Pods must queue at the workstation before being processed for picking. Figure 2 shows a system with pick stations along three sides. Replenishment stations are similar to pick stations, but products are transferred from incoming pallets, roll cages, or totes to the storage pods. Section 6 investigates a case with both pick and replenishment stations.

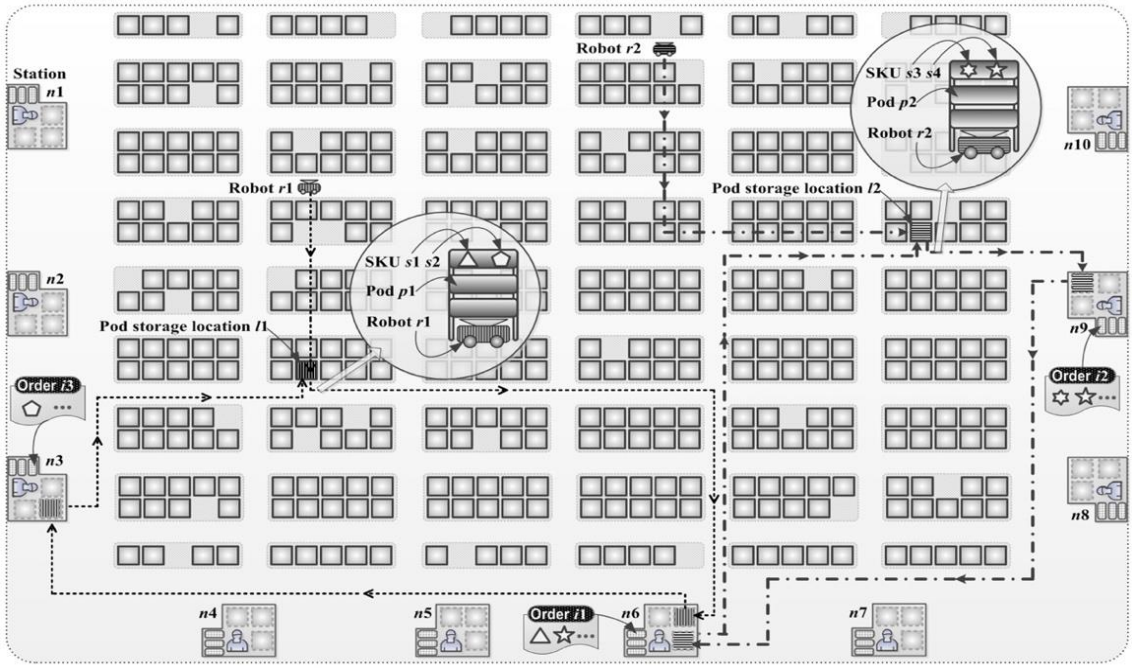


Figure 2: Routes of robots and pods for fulfilling orders at pick stations

Figure 2 shows the travel paths (dashed lines) of two robots and the pods they carry. Robot $r1$ goes from its current location to Pod $p1$'s storage location, picks up $p1$, and then visits Station $n6$ and Station $n3$ sequentially because the orders handled at these two stations require the SKUs that are stored in Pod $p1$. After picking, Robot $r1$ carries Pod $p1$ back to $p1$'s original storage location. The transport activities of Robot $r2$ and Pod $p2$ are similar. Many RMFSs (e.g., Montapacking, a logistics service provider that runs operations for 60 e-commerce retailers in Europe) return pods to their original location, as in the example in Figure 2. However, pods do not need to be returned to their original locations when using a dynamic pod storage strategy, which may improve performance depending on subsequent order needs.

We assume orders arrive continuously. Once a certain number of orders have accumulated in the order queue, they are assigned to the pick stations. In the example in Figure 2, there are ten pick stations, 30 orders, and each pick station handles three orders concurrently. Before the decision system can assign the 30 orders to these ten stations, it needs to collect information about the pods' locations, which SKUs are stored in each pod and in which quantities, and the current locations and status (empty, laden, battery level, etc.) of the robots. In addition, the system also needs related configuration parameters such as the travel time (distance) of each link in the network, consisting of arcs connecting the vertices (pod storage locations, station locations, and robot

locations). The decision system uses the related data, and solves the mathematical models to timely issue commands on the assignment of orders to stations, the pods required, their routes to the stations, the robots required, and their routes to their assigned pods. These decisions ensure that the pick work for all orders is completed as quickly as possible. After issuing the instructions, the decision system moves to the next batch of orders.

We analyze the system based on the following assumptions.

(1) Sufficient robots are available for the current set of orders, so they seldom create a bottleneck in the system. Thus, we do not plan multiple trips for each robot in the planning horizon for one batch of orders.

(2) The storage locations of pods, the SKUs and their quantities in each pod, the robots' locations, and the travel time between any pair of two locations are given as known parameters.

(3) While robot waiting at the pick stations is included, additional robot blocking and congestion in traveling between locations is not considered in the model. Since the robots can travel underneath the loads while empty and since multiple alternative Manhattan trajectories can be followed while loaded, the impact of congestion in the travel is typically moderate. We do investigate the impact of this assumption in Section 5.3.7.

These assumptions can be relaxed when needed, but they are close to current practice in many systems (e.g., at Montapacking, the company mentioned earlier).

4. Two decision methods for fulfilling a batch of orders

This section proposes two decision methods to handle a batch of orders (the 'batch mode'). One is a fast and practical method, suitable for large-scale applications. The other is an integrated method that can obtain better scheduling plans and solve large-scale instances. Both methods have relative advantages and are discussed in Sections 4.1 and 4.2, respectively. Section 4.3 elaborates the algorithm for solving the models embedded in the above methods.

4.1 A fast decision method based on a three-stage framework

The first decision method divides the complex decision problem into three stages. The decisions in each stage are optimized individually. The three stages are connected as the output of the first stage forms the inputs for the second and third stages. Note that the first decision method does not consider pod waiting time at the pick stations. It also uses a simplified expression to calculate the pick time. Figure 3 outlines the three-stage decision framework and shows an example of assigning a batch of 15 orders to five pick stations. Each station processes three orders.

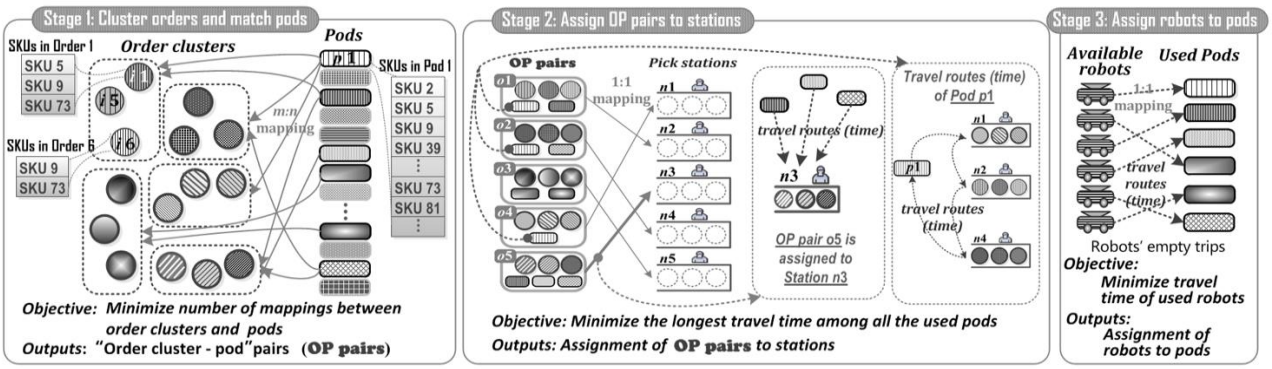


Figure 3: Three-stage decision framework for fulfilling a batch of orders

In the first stage, the orders are clustered into five groups of three orders each. It is efficient to pick products from different orders together (Gzara et al., 2020). The clustering criteria are based on the similarity of the orders, and orders with identical SKUs are clustered. In this case, the number of order clusters equals the number of stations. If the number of orders is not a multiple of the number of stations (e.g., assigning eight orders to three stations), it can differ by a maximum of one (i.e., stations process three, three, and two orders, respectively). Matching an order cluster with one or several pods produces order cluster-pod (OP) pairs. The quantity of a SKU for an order cluster (an order) may be satisfied by multiple pods, which implies we need not assume there is sufficient stock of a SKU in one pod for an order.

An OP pair is a pair of an order cluster and a set of pods. The goods in the set of pods are used to satisfy the demands of the orders in the cluster. In the second stage, the five OP pairs are assigned to five pick stations. The mapping between the pods and stations is many-to-many. The assignment of OP pairs to stations and the sequence in which each pod visits the stations are arranged to minimize the longest travel time (distance) among all the used pods. In the third stage, an available robot is assigned to each pod. As the robots' laden trips are optimized in the second stage, the objective in the third stage is to minimize the total empty travel time (distance) from the robots' current locations to the pods that they will pick up.

(1) Stage 1 model: clustering orders and matching pods

During the first stage, orders are clustered according to their SKU similarity to reduce the number of pods required and increase system efficiency. The model decides which orders should be clustered and which pods should be assigned to fulfill each order cluster. The variables are defined as follows:

Indices and sets:

i	index of an order	I	set of all orders
p	index of a pod	P	set of all pods
o	index of an order cluster	O	set of all order clusters
s	index of an SKU	S	set of all SKUs

Parameters:

f_{is} number of units of SKU s required by Order i

k_{ps} number of units in SKU s stored in Pod p

M a sufficiently large positive number

Decision variables:

ψ_{oi} binary, equals 1 if Order i is included in Order cluster o, and 0 otherwise

ξ_{op} binary, equals 1 if Order cluster o is fulfilled by Pod p, and 0 otherwise

θ_{ops} number of units of SKU s in Pod p picked to fulfill Order cluster o

Mathematical model:

[Submodel 1] *Minimize* $\sum_{o\in O}\sum_{p\in P}\xi_{op}$ (1)

s.t. $\lceil |I|/|O| \rceil - 1 \le \sum_{i\in I}\psi_{oi} \le \lceil |I|/|O| \rceil$ $\forall o \in O$ (2)

$\sum_{o\in O}\psi_{oi} = 1$ $\forall i \in I$ (3)

$\sum_{p\in P}\theta_{ops} = \sum_{i\in I} f_{is}\psi_{oi}$ $\forall o \in O, s \in S$ (4)

$\theta_{ops} \le M\xi_{op}$ $\forall o \in O, p \in P, s \in S$ (5)

$$\sum_{o\in O}\theta_{ops} \le k_{ps} \qquad \forall p \in P, s \in S \qquad (6)$$

$$\sum_{i\in I} i\psi_{oi} \le \sum_{i\in I} i\psi_{(o+1),i} \qquad \forall o \in O/\{|O|\} \qquad (7)$$

$$\psi_{oi} \in \{0,1\} \qquad \forall o \in O, i \in I \qquad (8)$$

$$\xi_{op} \in \{0,1\} \qquad \forall o \in O, p \in P \qquad (9)$$

$$\theta_{ops} \ge 0 \qquad \forall o \in O, p \in P, s \in S \qquad (10)$$

Objective (1) minimizes the number of the mappings between order clusters and pods, which implies the times that the used pods visit the stations. Constraints (2) and (3) state that $|I|$ orders are grouped into $|O|$ order clusters, and $\lceil |I|/|O| \rceil$ or $\lceil |I|/|O| \rceil - 1$ orders are included in each order cluster. Constraint (4) ensures that all SKUs required in an order cluster can be fulfilled by the pods assigned to that order cluster. Constraint (5) connects the two decision variables related to the assignment of pod p to Order cluster o. Constraint (6) guarantees that the number of units taken from each SKU in a pod does not exceed the number of SKU units in the pod. Constraint (7) reduces the symmetry of the solutions. Constraints (8)~(10) define the decision variables.

(2) Stage 2 model: assigning OP pairs to stations and pod routing

During the second stage, the model assigns the order clusters to pick stations, considering the travel time between the pods serving an order cluster and the station to which the order cluster is assigned. The model also determines the sequence in which pods visit pick stations, i.e., pod routing.

Input parameters from Stage 1 model:

ξ_{op} equals 1 if Order cluster o is fulfilled by Pod p, and 0 otherwise

Newly defined indices and sets:

n index of a pick station N set of all the pick stations

$e(p)$, $e'(p)$ index of a dummy location denoting the start point and end point, respectively, of Pod p's travel route. The start point can be regarded as a dummy location that is sufficiently close to the first station visited by Pod p, and the same for the end point. Note that $e(p)$ and $e'(p)$ can be but are not necessarily identical. The latter implies that the pod's storage positions can change dynamically.

Newly defined parameters:

t^1_{pn} travel time between Pod p's storage location and Station n

$t^2_{nn'}$ travel time between Station n and Station n'

Newly defined decision variables:

β_{on} binary, equals 1 if Order cluster o is handled at Station n, and 0 otherwise

ϑ_{pn} binary, equals 1 if Pod p visits Station n, and 0 otherwise

κ_p binary, equals 1 if Pod p is used, and 0 otherwise

$\delta_{pnn'}$ binary, equals 1 if Pod p visits Station n and Station n' sequentially, and 0 otherwise

$\bar{\delta}_{pn}$ position of Station n in Pod p's tour (i.e., sequence of stations visited by Pod p)

Mathematical model:

[Submodel 2] *Minimize* $Max_{p\in P}\{\sum_{n\in N} t^1_{pn}\delta_{p,e(p),n} + \sum_{n\in N} t^1_{pn}\delta_{p,n,e\prime(p)} + \sum_{n\in N}\sum_{n'\in N} t^2_{nn'}\delta_{pnn'}\}$ (11)

s.t. $\sum_{n\in N}\beta_{on} = 1 \qquad \forall o \in O \qquad (12)$

$$\sum_{o\in O}\beta_{on}=1 \qquad \forall n\in N \tag{13}$$

$$\sum_{o\in O}\xi_{op}\beta_{on}=\vartheta_{pn} \qquad \forall n\in N, p\in P \tag{14}$$

$$\xi_{op}\le\kappa_p \qquad \forall o\in O, p\in P \tag{15}$$

$$\vartheta_{pn}\le\kappa_p \qquad \forall n\in N, p\in P \tag{16}$$

$$\sum_{n'\in N\cup\{e'(p)\}}\delta_{pnn'}=\sum_{n'\in N\cup\{e(p)\}}\delta_{pn'n}=\vartheta_{pn} \qquad \forall p\in P, n\in N \tag{17}$$

$$\sum_{n\in N}\delta_{p,n,e'(p)}=\kappa_p \qquad \forall p\in P \tag{18}$$

$$\sum_{n\in N}\delta_{p,e(p),n}=\kappa_p \qquad \forall p\in P \tag{19}$$

$$\bar{\delta}_{pn'}\ge\bar{\delta}_{pn}+1-M\left(1-\delta_{pnn'}\right) \qquad \forall p\in P, n\in N\cup\{e(p)\},\ n'\in N\cup\{e'(p)\} \tag{20}$$

$$\beta_{on}\in\{0,1\} \qquad \forall o\in O, n\in N \tag{21}$$

$$\vartheta_{pn}\in\{0,1\} \qquad \forall p\in P, n\in N \tag{22}$$

$$\kappa_p\in\{0,1\} \qquad \forall p\in P \tag{23}$$

$$\delta_{pnn'}\in\{0,1\} \qquad \forall p\in P, n\in N\cup\{e(p)\},\ n'\in N\cup\{e'(p)\} \tag{24}$$

$$\bar{\delta}_{pn}\in Z^+ \qquad \forall p\in P, n\in N\cup\{e(p),e'(p)\} \tag{25}$$

Objective (11) minimizes the longest travel time among all used pods. Constraints (12) and (13) ensure that each order cluster is assigned to only one pick station and that each station handles only one order cluster. Constraint (14) connects the two assignment decisions, i.e., assigning order clusters to stations and pods to stations. Constraint (15) derives the variable κ_p that denotes whether a pod is used or not. Constraint (16) ensures that when Pod p is not used, ϑ_{pn} is 0. Constraints (17)~(19) construct the travel route for each pod used. Constraint (20) eliminates possible subtours according to the idea proposed by Miller et al. (1960); here the "tour" is the sequence of stations that a pod visits. Constraints (21)~(25) define the decision variables.

(3) Stage 3 model: assigning robots to pods

During the third stage, a robot is assigned to each pod. As the sequence of stations that a pod visits has been determined, the decision is based on minimizing the travel time between a robot's original location and the storage location of the pod it is carrying.

Input parameters from Stage 1 model:

κ_p equals 1 if Pod p is used, and 0 otherwise

Newly defined indices and sets:

r index of a robot R set of all available robots

Newly defined parameters:

t^3_{rp} travel time between Robot r's current location and Pod p's storage location

Newly defined decision variables:

α_{rp} binary, equals 1 if Robot r is assigned to carry Pod p, and 0 otherwise

Mathematical model:

[Submodel 3] *Minimize* $\sum_{r\in R}\sum_{p\in P'}t^3_{rp}\alpha_{rp}$ (26)

s.t. $\sum_{p\in P}\alpha_{rp}\le 1$ $\forall r\in R$ (27)

$$\sum_{r\in R}\alpha_{rp}=\kappa_p \qquad \forall p\in P \tag{28}$$

$$\alpha_{rp}\in\{0,1\} \qquad \forall r\in R, p\in P \tag{29}$$

Objective (26) minimizes the total time the robots travel empty from their current locations to the storage locations of the pods that they carry. Constraints (27) and (28) ensure that each pod is carried by only one robot and that a robot carries only one pod. Constraint (29) defines the decision variables.

Solving the models of the decision framework is not an easy task because both submodels 1 and 2 are NP-hard (Garey and Johnson, 1979; Dawande et al., 2008). Submodel 3 is an unbalanced assignment problem that can be solved in polynomial time.

Proposition 1: Submodels 1 and 2 are NP-hard.

Proof: See Appendix A. ■

4.2 An integrated decision method

This subsection presents an integrated model to optimize decisions in the system. These decisions include assigning orders to stations, pods to stations, robots to pods, and the sequence in which each pod visits the stations. The objective of the integrated model is to finish all the orders in a batch as quickly as possible. Figure 4 illustrates the activities and intertwined decisions with three pick stations, five robots, and five pods. Each line illustrates the movement of a pod to two or three stations in sequence and to its storage location after picking operations.

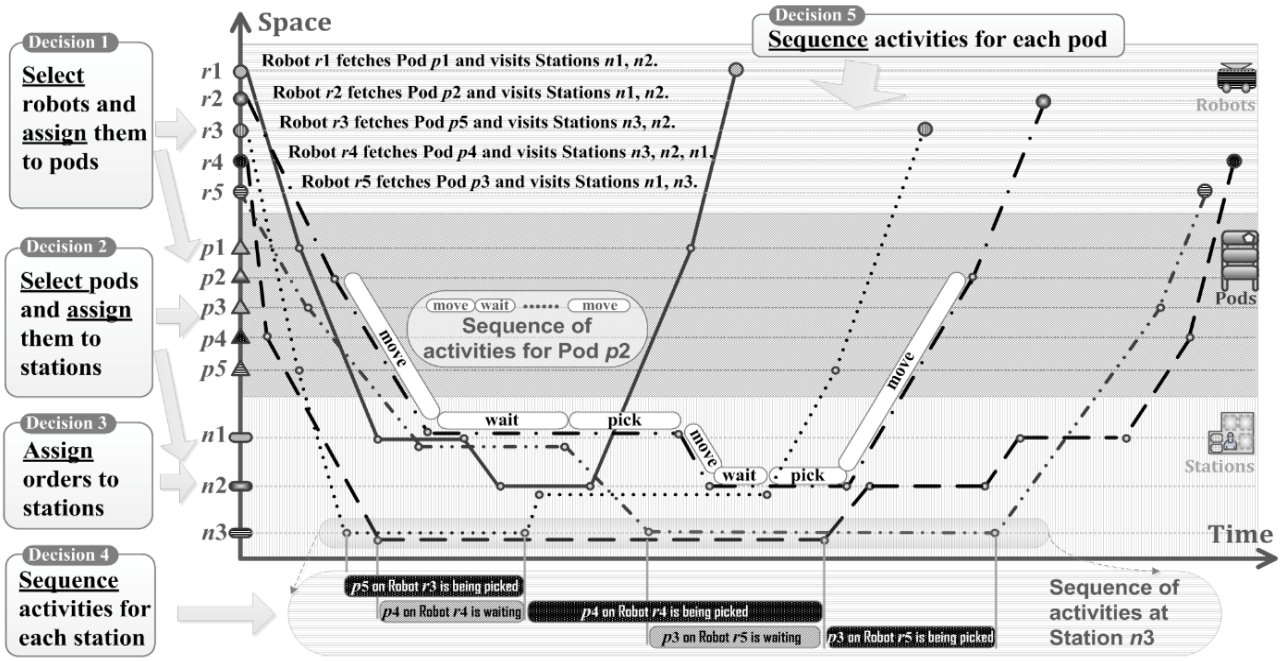


Figure 4: Intertwined decisions for an example with three pick stations

The model proposed in this section differs significantly from (and is more challenging than) the existing scheduling or routing problems such as vehicle routing problems (VRPs), although some constraints have a structure similar to that of a VRP. The integrated model is formulated as follows.

Newly defined indices and sets:

$\dot{e}(n)$, $\dot{e}'(n)$ index of a dummy first and last pod, respectively, of a sequence of pods visiting Station n

G set of unit sizes, indexed by g; g=1, 2, 3 denotes small, medium, and large, respectively

Newly defined parameters:

l_{sg} equals 1 if the unit size of SKU s has value g, and 0 otherwise

$Q(s,a)$ function estimating the handling time for picking a units from SKU s. The time depends on the

size of the product units. When a picker picks small-sized units from a pod, only one pick motion is needed regardless of the number of units, and the duration of a pick motion is assumed to be one second. When picking medium-sized units, the picker can pick a unit with each hand in one pick motion, and the time taken to pick a units is $\lceil a/2 \rceil$. When picking large-sized units, the picker uses both hands to hold the unit in one slow pick motion lasting about two seconds, and the picking time for a units is $2a$. The function $Q(s,a)$ is as follows:

$$Q(s,a) = \begin{cases} min\{a,1\} & if\ \ l_{s1} = 1 \\ \lceil a/2 \rceil & if\ \ l_{s2} = 1 \\ 2a & if\ \ l_{s3} = 1 \end{cases}$$

The handling time depends on the number of picked units of an SKU and the number of SKUs. Therefore, we define the following handling time parameter for the number of picked SKUs.

h fixed time to pick one SKU from a pod at a station

For example, assume an order is handled at a station and needs two SKUs from a pod waiting at the station, one of which is small and the other medium-sized. The demand for the small-sized SKU is a units, and the demand for the medium-sized SKU is a' units. The pod will then wait at the station for a duration of $2h + min\{a,1\} + \lceil a'/2 \rceil$, corresponding to the handling time of picking $a + a'$ units of the two SKUs.

Newly defined decision variables:

φ_{pn} Pod p's arrival time at Station n

ρ_{pn} Pod p's start time for picking SKUs at Station n

μ_{ops} binary, equals 1 if SKU s in Pod p is picked for Order cluster o, and 0 otherwise

γ_{pn} number of SKUs in Pod p picked at Station n

$\sigma_{npp'}$ binary, equals 1 if Pod p and Pod p' visit Station n sequentially, and 0 otherwise

$\bar{\sigma}_{np}$ position of Pod p in the sequence of pods that visit Station n

Mathematical model:

$$[\boldsymbol{M_{Batch}}] \quad Minimize \quad max_{n\in N}\{\rho_{\dot{e}'(n),n}\} \tag{30}$$

s.t. Constraints (2)~(10); (12)~(25); (27)~(29).

$$\theta_{ops} \le M\mu_{ops} \qquad \forall o \in O, p \in P, s \in S \tag{31}$$

$$\gamma_{pn} = \sum_{o\in O} \beta_{on} \sum_{s\in S} \mu_{ops} \qquad \forall p \in P, n \in N \tag{32}$$

$$\rho_{pn} + h\gamma_{pn} + \sum_{s\in S} Q\left(s, \sum_{o\in O} \beta_{on}\theta_{ops}\right) + t^2_{nn'} \le \varphi_{pn'} + M\left(1 - \delta_{pnn'}\right) \qquad \forall p \in P, n \in N \cup \{e(p)\}, n' \in N \cup \{e'(p)\} \tag{33}$$

$$\sum_{r\in R} t^3_{rp}\alpha_{rp} + t^1_{pn} \le \varphi_{p,n} + M\left(1 - \delta_{p,e(p),n}\right) \qquad \forall p \in P, n \in N \tag{34}$$

$$\varphi_{pn} \le \rho_{pn} \qquad \forall p \in P, n \in N \tag{35}$$

$$\rho_{pn} + h\gamma_{pn} + \sum_{s\in S} Q\left(s, \sum_{o\in O} \beta_{on}\theta_{ops}\right) \le \rho_{p'n} + M\left(1 - \sigma_{npp'}\right) \qquad \forall n \in N, p \in P \cup \{\dot{e}(n)\}, p' \in P \cup \{\dot{e}'(n)\} \tag{36}$$

$$\sum_{p'\in P\cup\{\dot{e}'(n)\}} \sigma_{npp'} = \sum_{p'\in P\cup\{\dot{e}(n)\}} \sigma_{np'p} = \vartheta_{pn} \qquad \forall p \in P, n \in N \tag{37}$$

$$\sum_{p\in P} \sigma_{n,p,\dot{e}'(n)} = 1 \qquad \forall n \in N \tag{38}$$

$$\sum_{p\in P} \sigma_{n,\dot{e}(n),p} = 1 \qquad \forall n \in N \tag{39}$$

$$\bar{\sigma}_{np'} \geq \bar{\sigma}_{np} + 1 - M\left(1 - \sigma_{npp'}\right) \qquad \forall n \in N, p \in P \cup \{\dot{e}(n)\}, p' \in P \cup \{\dot{e}'(n)\} \qquad (40)$$

$$\varphi_{pn}, \rho_{pn}, \gamma_{pn} \geq 0 \qquad \forall n \in N, p \in P \cup \{\dot{e}'(n)\} \qquad (41)$$

$$\mu_{ops} \in \{0,1\} \qquad \forall o \in O, p \in P, s \in S \qquad (42)$$

$$\sigma_{npp'} \in \{0,1\} \qquad \forall n \in N, p \in P \cup \{\dot{e}(n)\}, p' \in P \cup \{\dot{e}'(n)\} \qquad (43)$$

$$\bar{\sigma}_{np} \in Z^{+} \qquad \forall n \in N, p \in P \cup \{\dot{e}(n), \dot{e}'(n)\} \qquad (44)$$

Objective (30) minimizes the time to complete all orders at the stations. The purpose of Constraints (31) and (32) is to calculate the number of SKUs picked from Pod p at Station n (i.e., γ_{pn}), which is then used to derive the handling time. Constraint (33) connects a pod's start handling time at a station (i.e., ρ_{pn}) and the pod's arrival time at its next station (i.e., $\varphi_{pn'}$). The time interval includes the handling time and the travel time between the two stations (i.e., $t^2_{nn'}$). Constraint (34) determines the first pod's arrival time at each station, based on the travel time of the robot assigned to the pod from its current location to the pod's storage location (i.e., $\sum_{r\in R} t^3_{rp}\alpha_{rp}$), and the travel time from the pod's storage location to the pick station (i.e., t^1_{pn}). Constraint (35) ensures that a pod's start handling time at a station is after the pod's arrival time at the station. Constraint (36) connects a pod's start handling time at a station (i.e., ρ_{pn}) and the next pod's start handling time at the station (i.e., $\rho_{p'n}$); the time interval is also the handling time as mentioned above (i.e., $h\gamma_{pn} + \sum_{s\in S} Q(s, \sum_{o\in O}\beta_{on}\theta_{ops})$). Constraints (37)~(39) construct the sequence of pods visiting each station. Constraint (40) eliminates possible subtours (here "tour" is the sequence of pods that visit a station, which is different from the concept of tour in Constraint (20)). Constraints (41)~(44) define newly added variables.

Constraints (16), (32), (33), and (36) are nonlinear as they contain the product of two variables. They are linearized by adding some variables and constraints. Appendix B describes the details.

Proposition 2: A lower bound of the model M_{Batch} is: $LB = \min_{r\in R, p\in P', n\in N}(t^1_{pn} + t^3_{rp}) + \frac{\sum_{s\in S'} Q(s, \sum_{i\in I} f_{is}) + h|S'|}{|N|}$; here $P' = \{p | p \in P \text{ and } \sum_{i\in I}\sum_{s\in S} f_{is}k_{ps} > 0\}$ and $S' = \{s | s \in S \text{ and } \sum_{i\in I} f_{is} > 0\}$.

Proof: See Appendix C. ■

To accelerate the solution process, we use the lower bound proposed in Proposition 2 to formulate some other lower bounds of the submodels, which are embedded in the algorithm for solving the model M_{Batch}.

4.3 Algorithm for solving model M_{Batch}

This subsection elaborates a tailored metaheuristic algorithm to solve the integrated model M_{Batch} for large-scale cases in a short computation time. The metaheuristic is called the "Two-layer Revolving Algorithm" (TRA). It is a two-layer solution framework with iteratively fixed variables. Figure 5 illustrates the core idea of the TRA metaheuristic. This TRA metaheuristic has the following two features:

Some variables are fixed in a revolving manner under a two-layer framework: We divide the variables over multiple layers and temporarily and iteratively fix some of the variables, as done by Tseng and Yun (2009), Yun et al. (2011), and Zhen et al. (2011). Our proposed TRA fixes some variables in a revolving manner under a two-layer framework. Figure 5 shows two colored rings that denote the solution process within the two layers. A relaxed model (denoted by M^R_{Batch}) is solved in the inner gray layer and the original model (M_{Batch}) is solved in the outer blue layer. Appendix D presents details of model M^R_{Batch}. The solution

processes in the inner and outer layers are executed with some fixed variables to reduce solution space and thereby accelerate solution speed. As shown in Figure 5, two variables located under the gray ring and the nine variables inside the blue ring are fixed during the solution of the inner layer and the outer layer models, respectively. The three core variables (α, ϑ, ψ inside the inner circle) have a significant impact on the solution. They are fixed in a revolving manner, as the optimality improvement in solving the relaxed model is limited if all of them are fixed. Thus, we fix two 'gray' layer variables in each iteration. The revolving wheel in Figure 5 shows that three combinations of two variables are fixed across three iterations.

Some lower bounds are embedded in the metaheuristic: We use some lower bounds in the embedded procedures to judge the quality of the current solution in advance and accelerate the solution process of the metaheuristic. The lower bounds are calculated using the current solution values and then compared with the best objective value obtained so far. We determine specific decision variable values in each of the three sub-algorithms. Then, for each of the lower bound methods, we calculate and compare a lower bound with these values with the best objective value obtained so far. If the comparative gap is larger than a predefined threshold, it means the lower bound with these fixed values given is very small, which implies these values are potentially "good" (since we have a minimization problem). The remainder of the procedure will then be executed to determine the other decision variable values. Otherwise, the rest of the procedure will not be executed, thereby saving solution time. The lower bounds are used at the two layers of the metaheuristic in different ways. The used lower bounds are given in Proposition 3, which builds on the lower bound stated in Proposition 2.

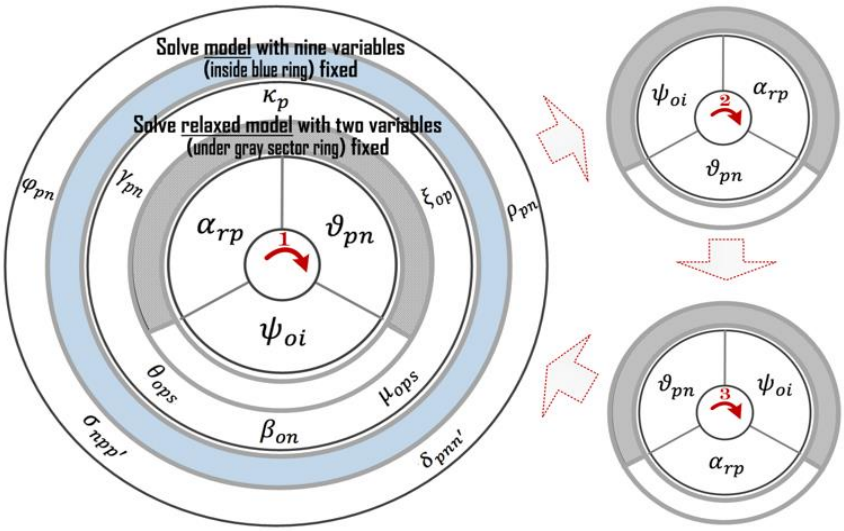


Figure 5: The core idea of the TRA metaheuristic

The TRA framework contains three procedures denoted by $\mathbb{F}_1(\boldsymbol{X})$, $\mathbb{F}_2(\boldsymbol{X})$, $\mathbb{F}_3(\boldsymbol{X})$. In each procedure (e.g., $\mathbb{F}_n(\boldsymbol{X})$), we first use a sub-algorithm (denoted by Sub-algorithm n) to solve the relaxed model M^R_{Batch} with variables $\boldsymbol{X}$ fixed at their values from the previous solution, and then solve the original model M_{Batch} with variables $\kappa_p, \alpha_{rp}, \vartheta_{pn}, \beta_{on}, \psi_{oi}, \xi_{op}, \theta_{ops}, \mu_{ops}, \gamma_{pn}$ fixed at their values from the previous solution (and hence the remaining four variables are not fixed). The resulting objective value is set to z. Appendix E presents details on Sub-algorithms 1, 2, and 3.

In the TRA metaheuristic, some variables are fixed in the sub-algorithms while solving the problem. To

accelerate the solution process, Proposition 3, which is based on Proposition 2, proposes three lower bounds for the model with three different combinations of the fixed variables. For example, as shown in Framework 1, given $\boldsymbol{\mathcal{X}} = (\boldsymbol{\alpha}, \boldsymbol{\vartheta})$, we judge the condition $(z^* - LB(\boldsymbol{\mathcal{X}}))/z^* > e$, which implies the gap between $LB(\boldsymbol{\alpha}, \boldsymbol{\vartheta})$ (i.e., the lower bound with $\boldsymbol{\alpha}$ and $\boldsymbol{\vartheta}$ given) and z^* (i.e., the best objective value obtained so far) is larger than a threshold e. If the condition holds, the procedure $\mathbb{F}_1(\boldsymbol{\alpha}, \boldsymbol{\vartheta})$ will be executed to determine the values of the remaining decision variables with $\boldsymbol{\alpha}$ and $\boldsymbol{\vartheta}$ given. Otherwise, the procedure will be skipped in this iteration to save computation time. In each procedure (sub-algorithm), the lower bounds are also used for acceleration (see Appendix E for details). Thus, these lower bounds are used in the two layers of the metaheuristic to accelerate the solution process.

Framework 1: TRA metaheuristic framework

$mark \leftarrow 0;\ stop \leftarrow 0;\ id \leftarrow 1;\ seq = \{\alpha_{rp}, \vartheta_{pn}, \psi_{oi}\}$
Obtain an initial solution; the objective value is z^*. // Appendix F shows details
While $(1 - stop)$
  Set $\boldsymbol{\mathcal{X}}$ = ($[id\%3]^{th}$, $[(id+1)\%3]^{th}$) variables in the seq.
  If $((z^* - LB(\boldsymbol{\mathcal{X}}))/z^* > e)$ // $LB(\boldsymbol{\mathcal{X}})$ is defined in Proposition 3; e is a threshold that is set in advance
    Execute procedure $\mathbb{F}_{id\%3}(\boldsymbol{\mathcal{X}})$, the solved objective value is z.
    // Execute one of three procedures $\mathbb{F}_1(\boldsymbol{\mathcal{X}})$, $\mathbb{F}_2(\boldsymbol{\mathcal{X}})$, $\mathbb{F}_3(\boldsymbol{\mathcal{X}})$ and the corresponding sub-algorithm (Sub-algorithm 1, 2, or 3) embedded in the above three procedures, respectively.
  End if
  If $(z < z^*)$
    $z^* \leftarrow z;\ mark \leftarrow 0.$
  Else
    $mark \leftarrow mark + 1.$
  End if
  $id \leftarrow id + 1.$
  If ($mark \geq 3$ or $id \geq 50$) //Conditions for terminating procedure
    $stop \leftarrow 1.$
  End if
End while
Output objective value z^* and solved values of decision variables

Proposition 3: $LB(\boldsymbol{\mathcal{X}})$ is a lower bound of the model M_{Batch} with variable $\boldsymbol{\mathcal{X}}$ fixed. The three LBs used in the three sub-algorithms are formulated as follows:

$$LB(\boldsymbol{\alpha}, \boldsymbol{\vartheta}) = \max_{r\in R, p\in P, \alpha_{rp}=1} \left[t^3_{rp} + TSP(p, N_p) + |N_p|(h + \min_{i\in I}\{t_i | t_i = \textstyle\sum_{s\in S} Q(s, f_{is}), t_i > 0\}) \right];$$

$$LB(\boldsymbol{\psi}, \boldsymbol{\alpha}) = \min_{r\in R, p\in P, \alpha_{rp}=1} (t^3_{rp} + \min_{n\in N} t^1_{pn}) + \min_{o\in O}[\textstyle\sum_{s\in S} Q(s, \sum_{i\in I} \psi_{oi} f_{is}) + h \sum_{i\in I} \psi_{oi} | \{s | s \in S, f_{is} > 0\}|];$$

$$LB(\boldsymbol{\vartheta}, \boldsymbol{\psi}) = \max\{lb1, lb2\};$$

here: $lb1 = \max_{p\in P''} \left[\min_{r\in R} t^3_{rp} + TSP(p, N_p) + |N_p|(h + \min_{i\in I}\{t_i | t_i = \textstyle\sum_{s\in S} Q(s, f_{is}), t_i > 0\}) \right],$

$lb2 = \min_{r\in R, p\in P''} (t^3_{rp} + \min_{n\in N} t^1_{pn}) + \min_{o\in O}[\textstyle\sum_{s\in S} Q(s, \sum_{i\in I} \psi_{oi} f_{is}) + h \sum_{i\in I} \psi_{oi} | \{s | s \in S, f_{is} > 0\}|],$

$N_p = \{n | n \in N \text{ and } \vartheta_{pn} = 1\},$

$TSP(p, N_p)$ is the objective value of a TSP for a set of vertices ($N_p \cup \{p\}$) with p as the origin,

$P'' = \{p | p \in P \text{ and } \sum_{n\in N} \vartheta_{pn} > 0\}.$

Proof: See Appendix G. ■

5. Numerical experiments

In this section, we conduct experiments to validate the efficiency of the proposed metaheuristic and the effectiveness of the proposed models. We also derive some managerial insights. The experiments were implemented and performed on a workstation with two Xeon E5-2643 V4 CPUs (12 cores) running at 3.4 GHz with 256 GB of memory under Windows 10. The models and submodels embedded in the metaheuristic were implemented with CPLEX 12.6.1 in concert with C# (VS2015).

5.1 Experiment settings

The data for the experiments are based on an RMFS warehouse in Huizhou City, Guangdong Province, China, with an area of about 5,000 m^2 (82 m × 61 m), which supports the CAINIAO™ logistics network owned by China's largest e-commerce company Alibaba. Figure 6 shows the aisle layout of the warehouse used in our experiments. The aisle width is two meters. There are six pods per block, with ten aisles along both the horizontal and vertical axis. This RMFS warehouse system has about 200 robots and stores about 22,000 SKUs, with about 600,000 product units on stock. It handles about 80,000 orders daily, with up to 120,000 daily orders during promotions. Each robot is 0.65m × 0.43m × 0.29m and can travel at about 1m/s. Each pod is 0.9m × 0.9m × 1.8m. Each pod has four levels and stores at most 24 SKUs. Each SKU in a pod has at most 25 units; thus, the parameter k_{ps} is generated by a discrete uniform distribution U[0, 25]. In this company, most e-commerce orders contain just a few lines, each in very small quantities. We assume that the demand per SKU s in order i, f_{is}, follows a discrete uniform distribution U[1, 5] and that the number of SKUs in an order follows a discrete distribution. The probability for one, two, and three lines in one order is 80%, 15%, and 5%, respectively, which corresponds to the order size distribution of the warehouse. The average number of lines (SKUs) in one order is therefore 1.25. The percentage of small, medium, and large-sized SKUs in orders is 50%, 30%, and 20%. Table 2 lists six groups of generated instances. Figure 6 shows the positions of the pick stations in these instance groups.

Table 2: Key parameter settings for different problem instance groups

Numbers of / Instance groups	Orders $\|I\|$	Stations $\|N\|$	SKUs $\|S\|$	Pods $\|P\|$	Robots $\|R\|$	Decision variables	Constraints
ISG1	10	2	8	5	5	4×10^2	2×10^3
ISG2	12	2	9	6	6	5×10^2	4×10^3
ISG3	50	4	38	20	20	9×10^3	7×10^5
ISG4	100	4	75	40	40	3×10^4	1×10^7
ISG5	280	10	210	120	120	7×10^5	2×10^9
ISG6	400	14	300	160	160	2×10^6	9×10^9

Notes: $|S|$ = the number of the SKUs required by the orders in the instances; $|P|$ = the number of pods that store the required SKUs (rather than the total number of all pods in the warehouse).

The instance setting in Table 2 is determined according to the following estimation. This warehouse contains about 14 pick stations and handles 80,000 orders every day (16 hours of operation). The average handling time of each order is about ten seconds, i.e., 14×16×3600÷80000≈10. If the system assigns orders to stations every five minutes, each station can handle at most 30 orders during this time, i.e., 5×60÷10=30. Therefore, the

ratio ($|I|/|N|$) in Table 2 is no greater than 30. We consider 400 orders in the largest-scale instance group (ISG 6) because the warehouse handles about 400 orders every five minutes. Although it stores more than 20,000 SKUs, the number of SKUs related to the 400 orders is 500 at most (if the SKUs contained in the 400 orders do not overlap). However, in practice, orders can share common SKUs, especially for large batch sizes. Therefore, $|S|$ is set to 300 in ISG 6 (the largest scale experiment). Since each pod stores at most 24 SKUs or 12.5 SKUs per pod on average, the minimum number of pods that store these SKUs is 24, i.e., 300÷12.5=24, and the maximum is 300. Therefore, the number of pods $|P|$ in ISG6 is set at 160, i.e., the average of these values. Although a large warehouse may contain many more pods, up to 160 pods are involved simultaneously in the largest instances with 400 orders. We also assume the number of usable robots to carry these 160 potentially used pods is 160 in ISG6. Figure 7 demonstrates the distribution of SKU dispersion over pods for the instances in ISG6. As the system assigns about 400 orders every five minutes, the computation time of the decision model should not exceed five minutes. Otherwise, the model would not be useful in practice.

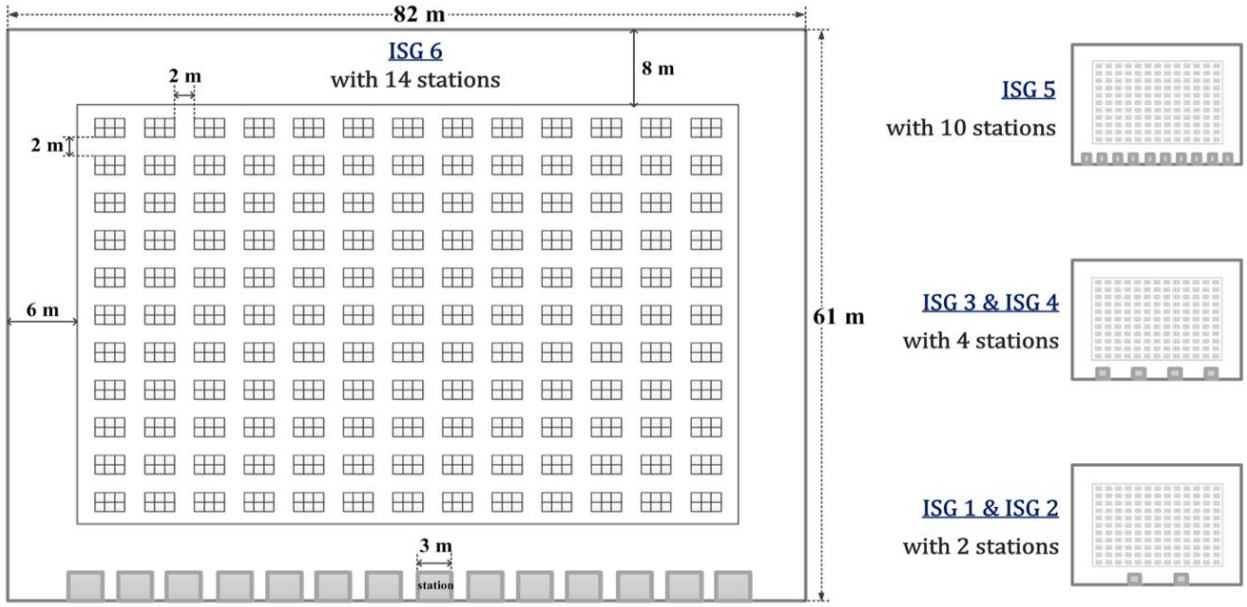


Figure 6: Warehouse configuration and layout for the six instance groups

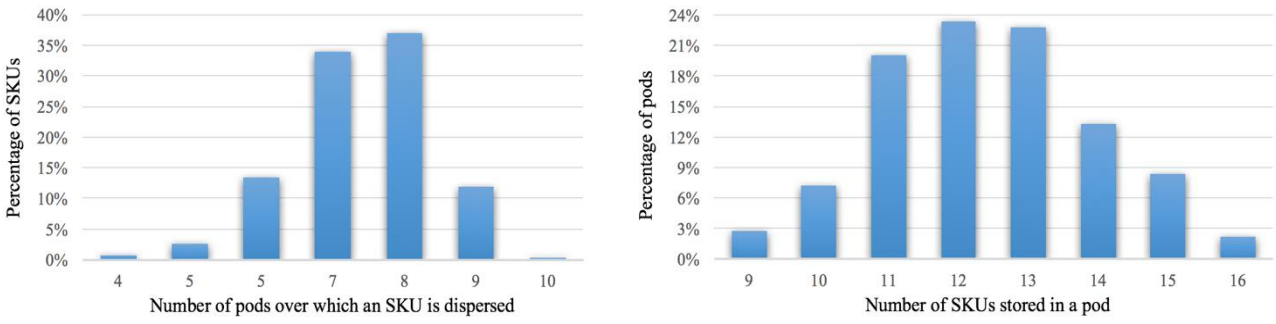


Figure 7: Distributions of SKU dispersion over pods (ISG 6)

5.2 Comparison of the fast decision method and the integrated decision method

This subsection compares the two proposed decision methods; both have relative advantages. We validate the performance of these methods relative to the optimality gap in some small-scale instances and compare the methods with some decision rules used in practice and proposed in other related work. To compare the two

methods meaningfully, we use Objective (30) to evaluate the solution objectives obtained by the two methods.

5.2.1 Comparing the two decision methods with optimal results in small-scale instances

The first series of experiments evaluate the performance of the two decision methods by comparing them with the optimal results obtained by CPLEX in small-scale instances. In Table 3, OBJ_{Cplex} denotes the optimal objective value for the integrated model, OBJ_{3stage} denotes the objective value of the three-stage fast decision method, and OBJ_{TRA} denotes the objective value for the integrated method, which is solved by our proposed solution method TRA. In order to find OBJ_{3stage}, the three-stage fast method is first used to obtain a solution on order-station assignment, robot-pod assignment, and pod routing along stations. Then, given this solution, the integrated model is used to calculate the objective value of the solution. This way of calculation allows a meaningful comparison between OBJ_{3stage} and the two objective values solved with the integrated model.

The results in Table 3 show that the TRA generates optimal (or near-optimal) results for ISG1 and ISG2 (see column Δ_{TRA}) with shorter computation times than CPLEX. CPLEX cannot solve ISG3 cases within two hours, but the TRA can obtain a solution quickly. This result validates the efficiency of the TRA for solving the model M_{Batch}. Column Δ_{3stage} also shows that the integrated decision method generates better results than the fast decision method, which sequentially solves three submodels.

Table 3: Comparison of decision methods in small-scale instances

Instance groups	#	**Optimal results**		**Fast method**	**Integrated method**		**Optimality gap**	
		OBJ_{Cplex}	Time (s)	OBJ_{3stage}	OBJ_{TRA}	Time (s)	Δ_{TRA}	Δ_{3stage}
ISG1	1	56.5	6	103.2	56.5	1	0%	83%
	2	74.2	5	113.2	74.2	1	0%	53%
	3	69.1	5	180.2	69.1	1	0%	161%
	4	48.2	5	100.2	48.2	1	0%	108%
	5	59.2	5	104.2	59.2	1	0%	76%
ISG2	1	52.4	49	128.2	52.4	1	0%	145%
	2	64.2	41	92.2	65.5	1	2%	44%
	3	71.5	141	146.2	72.5	1	1%	104%
	4	76.5	239	160.2	77.5	1	1%	109%
	5	59.5	93	66.1	60.5	1	2%	11%
ISG3	1	N/A	>7200	159.3	74.2	4	N/A	N/A
	2	N/A	>7200	123.3	61.9	8	N/A	N/A
	3	N/A	>7200	153.3	67.9	7	N/A	N/A
	4	N/A	>7200	163.3	88.1	25	N/A	N/A
	5	N/A	>7200	161.3	84.4	27	N/A	N/A
						Average:	1%	89%

Note: Δ_{TRA}=(OBJ_{TRA}−OBJ_{Cplex})/OBJ_{Cplex}; Δ_{3stage}=(OBJ_{3stage}−OBJ_{Cplex})/OBJ_{Cplex}.

5.2.2 Comparing the two decision methods with other decision rules in large-scale instances

We further validate the added value of the integrated decision method (based on M_{Batch} solved by the TRA) compared to the fast decision method inspired from practice and another decision rule (Merschformann et al., 2019) for large-scale instances. The TRA can solve the model for the largest scale instances with 14 stations, 400 orders, 300 SKUs, 160 pods, 160 robots within about four minutes, which is below the five-minute

threshold mentioned in Section 5.1. In Table 4, column $T1$ denotes the computation time of the TRA. The largest value in this column is 207 seconds, which is less than four minutes. As stated in Section 4.3, some lower bounds are used in the TRA. In Table 4, columns $T1$ and $T2$ denote the computation time with and without using these lower bounds, respectively. Column Δ_{Acc} denotes the acceleration obtained by using the lower bounds in the TRA. The average value is 28%, which validates the effectiveness of embedding the lower bounds in the algorithm.

We also compare the performance of our model with a decision rule from Merschformann et al. (2019), who use simulation to validate their 'nearest rule' (the best in pod selection) and the 'pod-match rule' (the best in order assignment) in their paper. The latter rule is similar to our three-stage fast decision method. We, therefore, adjust the fast decision method to incorporate the nearest rule. This rule assigns the pod that contains at least one required SKU and has the least travel time to the station (see Merschformann et al., 2019). Logistic service companies such as Montapacking Netherlands commonly use the 'nearest rule' in their RMFSs. We implement this decision rule by replacing submodel 2 (of Section 4.1) and selecting the pod that contains at least one SKU and has the least travel time to a station. The results are shown in Table 4. Note the large gap between the nearest decision rule and the fast decision method in some cases, reflecting the added value of submodel 2 compared to the nearest decision rule. The average value of Δ_{3stage}^{Rule} in Table 4 is about 22%, indicating that the fast decision method is better than a rule-based decision method. In addition, the average value of Δ_{TRA}^{Rule}, i.e., 73%, shows that the proposed model M_{Batch}, which is solved by the TRA, can also obtain a much better solution than the rule-based decision method.

Table 4: Comparison of decision methods in large-scale instances

Instance groups	#	**Nearest decision rule**	**Fast method**	**Integrated method**				**Gap from rule's results**	
		OBJ_{Rule}	OBJ_{3stage}	OBJ_{TRA}	$T1$(s)	$T2$(s)	Δ_{Acc}	Δ_{TRA}^{Rule}	Δ_{3stage}^{Rule}
ISG4	1	282.3	220.7	124.4	52	92	44%	127%	28%
	2	185.3	171.3	140.6	62	91	32%	32%	8%
	3	264.7	262.7	191.7	21	62	66%	38%	1%
	4	237.6	220.4	165.3	46	46	0%	39%	8%
	5	171.4	171.4	169.7	62	91	32%	1%	0%
ISG5	1	497.0	482.4	318.6	58	95	39%	56%	3%
	2	353.7	313.4	305.6	66	96	31%	16%	13%
	3	771.6	511.3	340.2	73	105	30%	127%	51%
	4	540.5	375.6	286.0	61	101	40%	89%	44%
	5	553.3	425.6	395.7	77	99	22%	40%	30%
ISG6	1	801.8	388.9	381.3	207	279	26%	110%	106%
	2	477.5	470.7	398.0	173	209	17%	20%	1%
	3	411.9	411.9	294.7	173	211	18%	40%	0%
	4	1279.3	960.1	361.3	164	199	18%	254%	33%
	5	660.4	633.6	334.0	185	205	10%	98%	4%
				Average:			28%	73%	22%

Note: $T1$ (and $T2$): the computation time with (and without) using the lower bounds; $\Delta_{Acc} = (T2 - T1)/T2$, denotes the acceleration rate by using the lower bounds; Δ_{TRA}^{Rule} = $(OBJ_{Rule} - OBJ_{TRA})/OBJ_{TRA}$; Δ_{3stage}^{Rule} = $(OBJ_{Rule} - OBJ_{3stage})/OBJ_{3stage}$.

We also conduct some experiments to test the effect of our initialization. This initialization is tailored to the problem (and elaborated in Appendix F). We compare it with the "nearest rule" initialization of Merschformann et al. (2019). We use our tailored heuristic and the nearest decision rule to obtain initial solutions and then apply the TRA to both initial solutions. Table 5 uses OBJ_T and OBJ_R to denote the final solution objective values obtained using the tailored heuristic and the nearest rule, respectively. The computation times resulting from the two initializations are denoted by $Time_T$ and $Time_R$. Here, the subscripts T and R denote the tailored heuristic and rule-based initialization, respectively. The gaps in the objective values and computation times are substantial (23% and 66%, respectively). This demonstrates the importance of having a good initial solution to improve algorithmic performance.

Table 5: Comparison of two initializations used in our proposed TRA metaheuristic

Instance groups	#	**Initialization by nearest decision rule**		**Initialization by our tailored method**		**Gap of objective values and computation time**	
		OBJ_R	$Time_R$ (s)	OBJ_T	$Time_T$ (s)	Δ_{R-T}^{OBJ}	Δ_{R-T}^{Time}
ISG5	1	485.3	604	318.6	58	34%	90%
	2	332.3	200	305.6	66	8%	67%
	3	478.2	426	340.2	73	29%	83%
	4	418.2	167	286.0	61	32%	63%
	5	459.4	228	395.7	77	14%	66%
					Average：	23%	66%

Note: $\Delta_{R-T}^{OBJ} = (OBJ_R - OBJ_T)/OBJ_R$; $\Delta_{R-T}^{Time} = (Time_R - Time_T)/Time_R$.

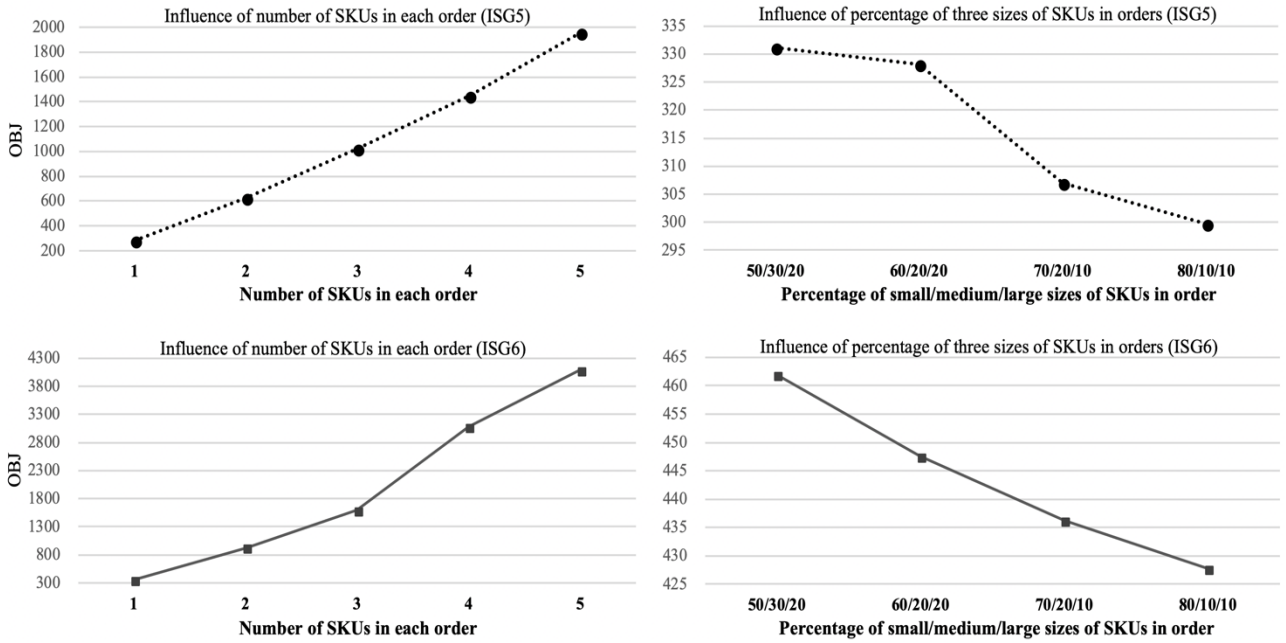
5.3 Impact of order composition and warehouse configurations

We conducted a sensitivity analysis to investigate the influence of order composition and warehouse operational policies on the makespan. Order composition refers to the number of SKUs in each order and their size (small, medium, or large), as the pick time may depend on this. The operational policies refer to short-term managerial decisions that influence performance, such as the number and position of pick stations, the SKU diversity in each pod, and the SKU dispersion over pods. This section only discusses the experimental results of ISG5 and ISG6 and their variations. All results in this subsection are average values measured over 20 randomly generated instances, solved by the TRA. For most instances, the half-widths of the 95% confidence intervals were below 10% of the mean, and the maximum half-widths were 15% of the mean.

5.3.1 Impact of order composition

We first investigate the influence of the number of SKUs in each order on performance. Unlike the basic setting, where the average number of SKUs in each order is 1.25, our sensitivity analysis assumes each order contains one, two, three, four, or five SKUs. We conduct five experiments for each problem instance group (i.e., ISG 5 and ISG 6). Figure 8(a) presents the experimental results of ISG 5 and ISG 6 instances. It shows that order completion time increases significantly with a higher average number of SKUs per order. As more SKUs need to be picked per order, more pods may be needed to travel to stations. The pods' travel time and waiting at stations increase the total order completion time. The above result may be predictable. If we divide the vertical by the horizontal coordinate value for each point in Figure 8(a), we find that the makespan per

SKU in each order increases with the number of SKUs per order. Merschformann et al. (2019) report a similar effect and mention the negative effect when larger orders must be handled, leading to more 'open SKUs' at the workstations. Figure 8(b) shows the impact of the percentage of small, medium, and large-sized SKUs in orders on the makespan. The higher the percentage of the small-sized SKUs in orders, the shorter the total completion time of handling all the orders. This is not only due to shorter picking times, but also to lower queuing times of the pods, which explains the nonlinear effects.



(a) (b)

Figure 8: Influence of (a) average number of SKUs in each order and (b) order composition on performance

5.3.2 Impact of pick station positions

RMFSs have great flexibility in deciding which pick stations are to be opened. However, the position of these pick stations impacts final performance (Lamballais et al., 2017). We investigate five layouts of pick stations, based on ISG5 with ten pick stations and ISG6 with 14 pick stations. Figure 9 shows ten or 14 pick stations that are open along one long side, three sides, two long sides, one short side, or two short sides of a rectangular warehouse. Stations are equidistantly distributed over the sides where they are positioned.

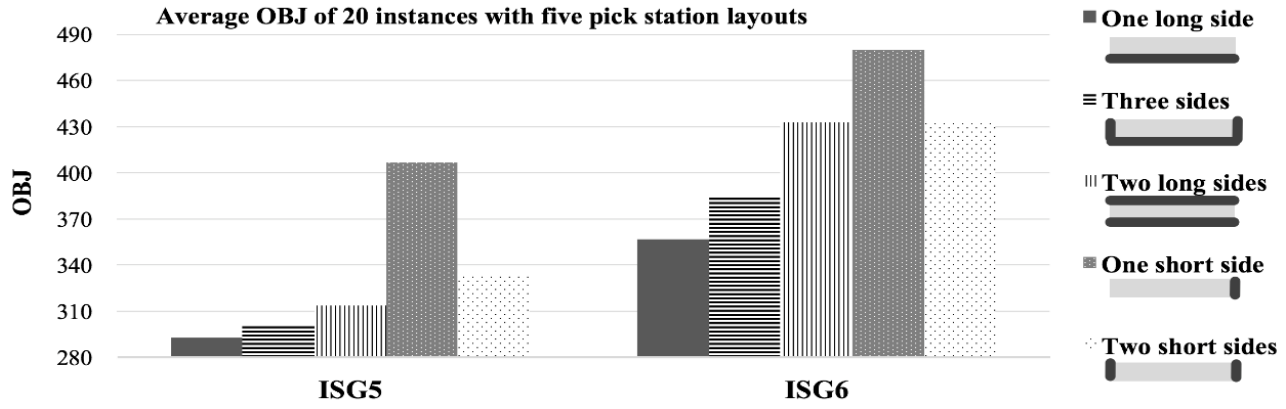


Figure 9: Influence of pick station positions on performance

Note that all results are averages, measured over 20 randomly generated instances. Figure 10 shows that the rank of each layout varies over the 20 instances. The layout with stations along one long side ranks first (best)

in 30% and 45% of the instances for ISG5 and ISG6, respectively. No layout is consistently best over all instances. However, positioning pick stations along one long side achieves the best performance in most instances. It reduces pod travel time as pods travel in the vertical direction (i.e., the short side of the warehouse) to a nearby station. Besides, using stations along the same side saves travel time for pods traveling between stations. These results are in line with those of Lamballais et al. (2017). They found that, for not zoned systems, throughput is highest when most pick stations are located along the long side of the warehouse.

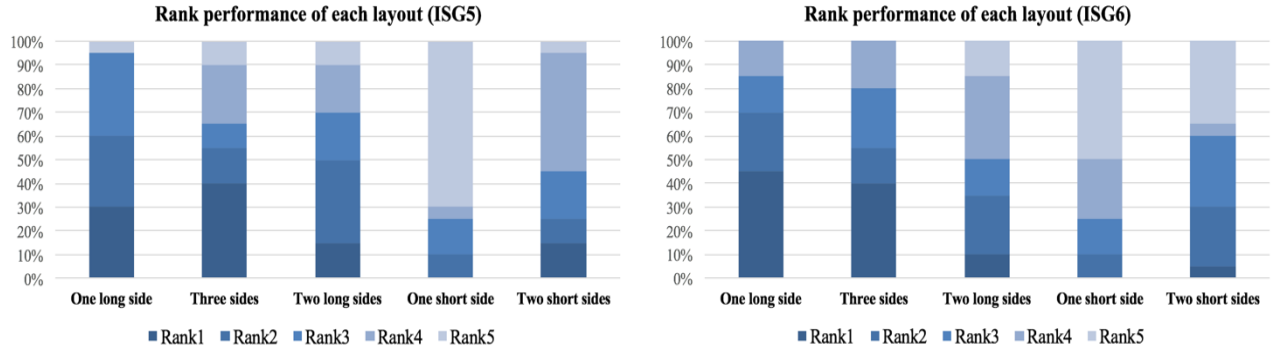


Figure 10: Rank performance (in %) per layout over 20 instances

5.3.3 Impact of the number of pick stations

Based on the best station position layout, i.e., along one long side, we further investigate the influence of the number of pick stations on the makespan. Except for the number of pick stations, all other parameters are the same as in ISG6. The number of pick stations varies between four and fourteen, evenly distributed over the long side. The orders, pods, and SKU distributions over pods are the same for the seven cases (based on ISG6). Figure 11 shows that the order completion time decreases as the number of stations increases. The reduction in work per station dominates the extra transport that might result from moving pods between stations. This result is in line with studies by Weidinger and Boysen (2018) and Wang, Yang, Qi (2022), although they did not include pod travel between stations. Note that these studies do not focus on order completion time (see Table 1) but on the number or travel time of pod moves. However, this measure is positively correlated with order completion time. Our result is also in line with the study of Yu and de Koster (2008), who also look at the trade-off between handling time and station transport time, for pick and pass systems, with a variable number of stations, (but with a fixed number of pickers). With a growing number of pick stations, the resulting mean order throughput time (which is correlated with makespan) decreases. However, when the number of pick stations becomes significantly larger, the mean order throughput time may increase slightly.

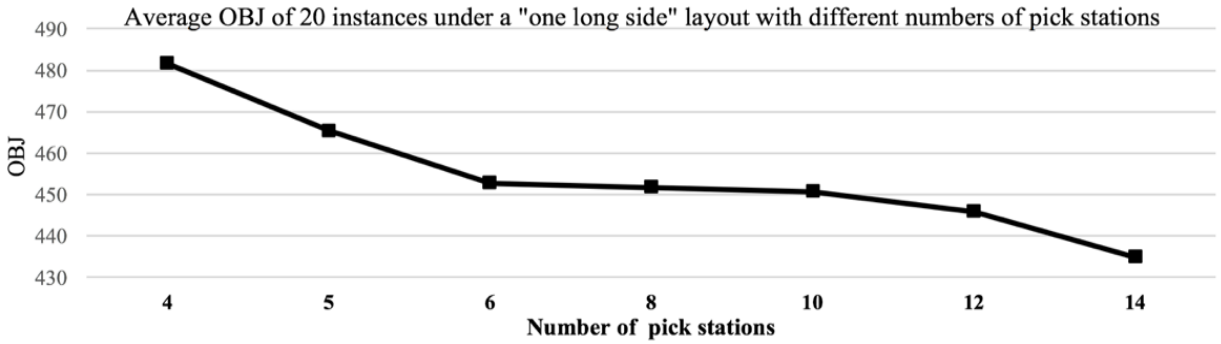
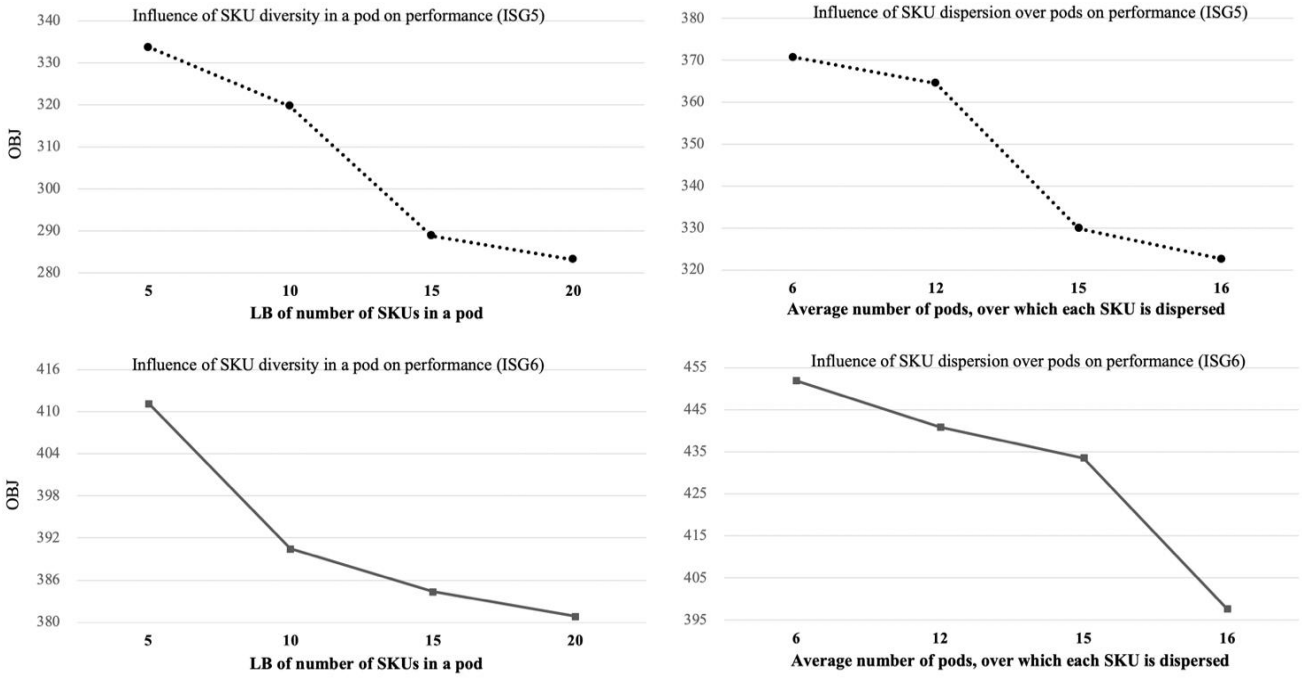


Figure 11: Influence of number of pick stations on performance

5.3.4 Impact of SKU diversity per pod

Shared storage, i.e., when SKUs do not have fixed storage positions but share common space, outperforms dedicated storage in terms of space consumption and retrieval time. The number of SKUs in each pod may also affect system performance. In the experimental setting, each pod stores at most 24 SKUs. We vary the minimum number of SKUs in each pod, denoted by LB on the horizontal axis in Figure 12(a). The results in Figure 12(a) show that the larger the LB, the greater the SKU diversity in each pod, and the greater the SKU diversity in each pod, the better the performance. With more SKUs per pod, fewer pods need to be fetched by the robots, and pods are more likely to be shared among stations. This result is in line with Wang, Yang, Qi (2022), who show that the more SKUs in each pod, the fewer pod moves are required to complete the orders. Hence, increasing the SKU diversity can effectively reduce the number of pod moves and achieve better makespan performance.



(a) (b)

Figure 12: Influence of (a) SKU diversity in a pod and (b) SKU dispersion over pods on makespan

5.3.5 Influence of SKU dispersion over pods

Next to having a high SKU diversity per pod, scattering SKUs over pods and locations also may help to increase performance (Goeke and Schneider, 2021). Many e-commerce retailers disperse (small) SKUs over multiple pods, increasing the chance that a robot can retrieve a close-by pod and thereby decreasing travel time (Weidinger and Boysen, 2018; Zhang et al., 2020). The study by Weidinger and Boysen (2018) shows that increasing the degree of 'scatteredness' of SKUs reduces the maximum picking distance per SKU. Zhang et al. (2020) use the term 'explosion ratio' to reflect SKU scatteredness. Our experiments use the average number of pods per SKU to reflect SKU dispersion over pods. We vary this average number of pods as 6, 12, 15, and 16, using the base instances of ISG5 and ISG6. First, we randomly generate an instance and then adjust the SKU dispersion iteratively so that the average number of pods per SKU reaches the target value of 6. Based on the instance with an average value of 6, we then randomly assign six more pods to each SKU to obtain the

instance with 12 pods per SKU. We continue in this way to obtain the other two instances. We repeat this process twenty times per instance. The results in Figure 12 are averages over these twenty values. Figure 12(b) clearly shows that a greater SKU dispersion over pods leads to better performance, i.e., shorter order completion time. These results are in line with the above two related studies. However, reducing order completion time may be offset by additional replenishment effort that may be required (depending on whether the pod replenishment process is concurrently executed on pods that are demanded in the pick process). In the next section, we describe how the model can be extended by including pod replenishment.

5.3.6 Impact of robot unavailability

This study assumes sufficient robots are available for the current set of orders (see the assumptions in Section 3). As the batch size is based on the number of available robots, it should be feasible to solve. However, the real-time information on the robots may be inexact; robots may have insufficient battery level, or a breakdown might occur. As a result, the number of actually available robots may be insufficient for the current batch and some robots may need to undertake more tasks to complete it. This will increase the makespan, i.e., the OBJ of the problem. We conduct experiments to investigate the influence of robot unavailability on makespan. For this, instance group ISG6 is used, with 160 available robots. The plan is based on these 160 robots; however, two, or more robots are actually unavailable. The change of the objective value under the above perturbations is calculated. The results in Figure 13 show the negative effect of unavailable robots on the order makespan. The effect becomes more pronounced with an increasing number of unavailable robots.

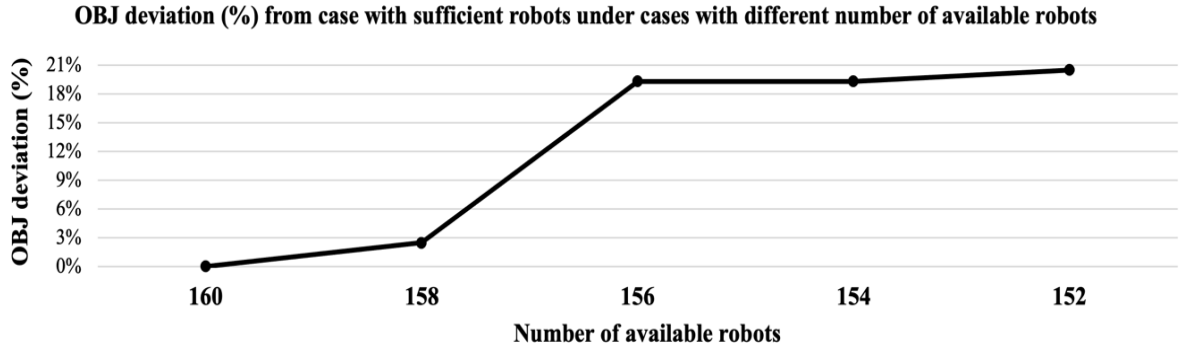


Figure 13: Influence of the number of unavailable robots on the order makespan

5.3.7 Robustness of the model against travel time deviations

This study also assumes the information on the storage locations of pods, the SKUs and their quantities in each pod, and the robot locations are exact, at the moment of calculating the plan. Indeed, the warehouse information systems keep track of these data in real time. However, particularly the travel time between any pair of locations may differ from the estimated value due to unforeseen traffic congestion or other factors in the warehouse operation. Therefore, robustness tests are conducted to investigate whether the performance is robust to the information on the travel time between any pair of two locations (i.e., parameters t^1_{pn}, $t^2_{nn'}$, and t^3_{rp}). The results in Figure 14 demonstrate that when the travel times are higher than their estimated values by 10%, 15%, …, 30%, the order makespan (i.e., OBJ) of the plan solved according to the estimated values is worse than the optimal result using the actual parameters by a percentage gap of less than 2%. Thus, the results in Figure 14 validate the robustness of the proposed model against travel time deviations.

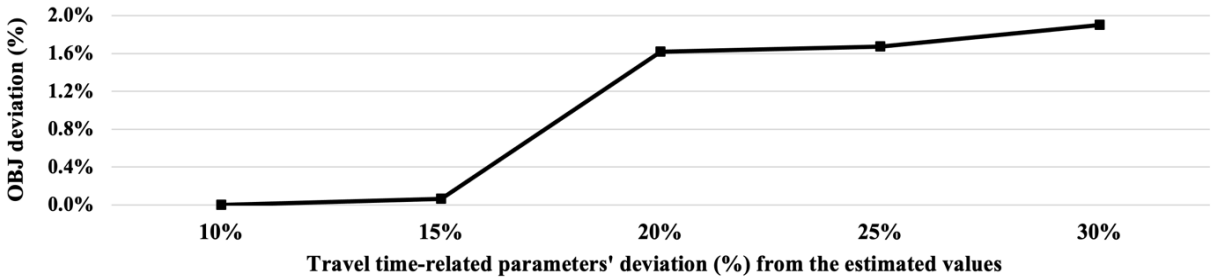


Figure 14: Robustness test of the travel time-related parameters

6. Three extensions

In this section, we investigate three extensions. First, we extend the batch mode model to a semi-batch mode model. Next, we extend the batch mode model to a multi-batch mode model. Last, we consider concurrent picking and pod replenishment in our model.

6.1 The semi-batch mode

The batch mode studied in the previous sections assumes that no orders are being handled at stations when assigning a batch of orders to a set of stations and pods. When all the orders in the batch are finished, the next batch of orders is assigned. In practice, new orders are assigned to pick stations frequently. However, some stations may already be processing orders, some pods may be waiting at the stations, and some may be on their way to the stations. The SKUs in the orders handled at a station and those stored on the pods waiting or on their way to a station can affect the assignment of new orders to a station (Merschformann et al., 2019). IoT-based technology can track the system status and capture information on the order handling process (Shi et al., 2021), the status of the pods, robots, stations, and SKUs on pods and at the stations, and put this into the RMFS scheduling system instantaneously. A decision model can then assign newly added orders to stations and schedule additional pods and robots.

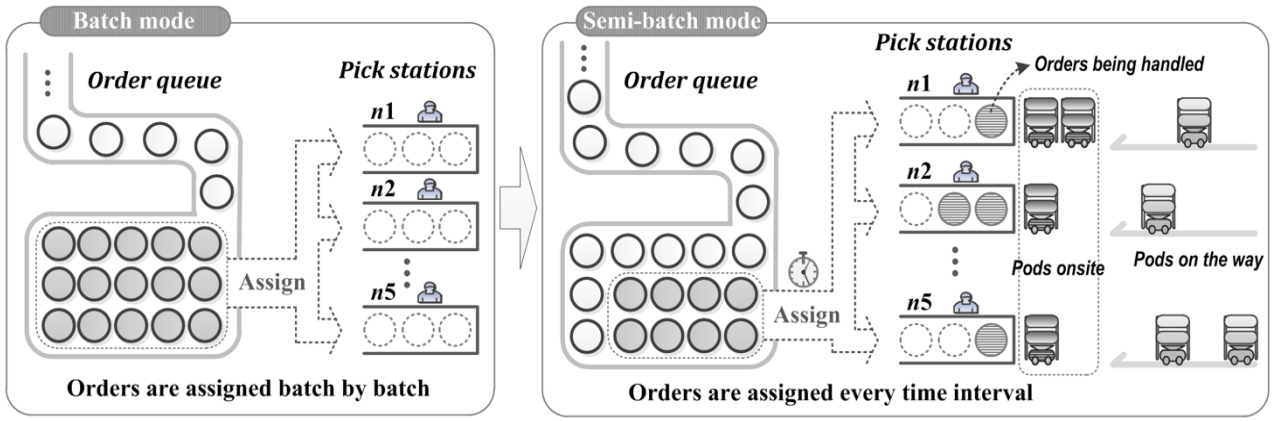


Figure 15: Batch mode and semi-batch mode for order fulfillment

We call this the *semi-batch* mode because the number of assigned orders is smaller than the full batch capacity of all the stations. As shown in Figure 15, the main difference between the semi-batch and the basic batch mode is that some orders are already being processed in the semi-batch mode, i.e., some robots are on their way to a station, and some have already taken pods to the station. The previous model M_{Batch} can also be used to handle the problem in the semi-batch mode, simply by fixing the decision variables related to the

orders being handled, the pods waiting at or on their way to a station, and the robots that carry these pods as known parameters. We can then simply solve the M_{Batch} model with partially fixed variables and obtain the assignment decisions for the newly added orders and the newly scheduled pods and robots.

In addition to running at a fixed frequency, the decision-making system can run the semi-batch mode at any time or even in the single-order flow mode, which means each batch contains only one order. Each order is then assigned to a station immediately when it arrives, instead of waiting for other orders to complete a batch of multiple orders to be assigned to a station.

Table 6: Comparison between the batch mode and the semi-batch mode

Instance groups	#	**Batch mode**		**Semi-batch mode**					
		OBJ	Time (s)	Every two minutes		Every 90 seconds		Every one minute	
				OBJ	Time (s)	OBJ	Time (s)	OBJ	Time (s)
ISG5	1	318.6	58	461.1	230	521.3	278	701.7	480
	2	305.6	66	496.5	182	659.3	219	683.3	350
	3	340.2	73	591.6	235	654.9	331	748.3	340
	4	286.0	61	525.0	220	622.3	327	705.4	433
	5	395.7	77	419.2	159	449.4	269	573.1	312
Average:		329.2	67	498.7	205	581.4	285	682.3	383

We conduct experiments to compare the batch and semi-batch modes using the same instance group ISG5. In the semi-batch mode, orders are dispatched at a fixed frequency. Table 6 shows three types of semi-batch modes. The batch is split into three, four, or five sub-batches, corresponding to sub-batch starting times of every two minutes, every 1.5 minutes, or every minute, respectively. In the semi-batch mode, current orders may already be in process at stations when orders are dispatched. The dispatch decision for each semi-batch is more complex than the normal batch mode because it considers ongoing orders in stations and pods waiting at or going to the stations. The batch mode performs significantly better than the semi-batch mode as it requires much shorter computation time. Comparing the three semi-batch modes shows that performance is related to the frequency at which orders are dispatched: the lower the frequency, the better the performance. This implies that the order dispatch frequency should not be set too high when this semi-batch mode is applied in real-life situations, as this can decrease optimization.

6.2 The multi-batch mode

The multi-batch mode applies to the context with many orders to handle, more than the number of robots, and all order information is known. For example, there are 4800 orders that are waiting to be picked and 160 robots available. Our algorithm cannot solve the case directly. The largest instances solvable are in ISG6, with 400 orders. Therefore, the above 4800 orders could be divided into twelve batches, each of 400 orders, solved batch by batch. Note that this multi-batch mode model differs from model M_{Batch} in one aspect. In a multi-batch mode the return locations of pods must be decided, after picking at the stations, because these locations may influence the makespan of the next batch of orders. Therefore, additional decision variables for the pod return storage locations must be added to model M_{Batch}. The pod return location decisions in a batch are input for the next batch's decision model. In addition, a constraint limiting the tour length is included (i.e. the number of pick stations visited in sequence), to avoid potential interference with the activities of the next batch. The

extended model for multi-batch mode is elaborated in Appendix I.

Experiments are conducted to validate the performance of solving these 4800 orders. The number of orders per batch is set as 400, 300, 200, or 100 orders, respectively. Here, the system performance is measured as the total length of the empty robot trips and the total length of the loaded robot trips (with pods). The results in Figure 16 (a) and (b) show that when the batch size decreases, the total length of the empty robot trips decreases, while the total length of the loaded robot trips increases first and then decreases. The results imply that the batch size should be set as a small value. However, Figure 16 (c) demonstrates that the computation efforts decrease for increasing batch sizes. Therefore, a proper batch size should be determined by balancing a plan's performance in trip length and the computational effort required for obtaining the plan.

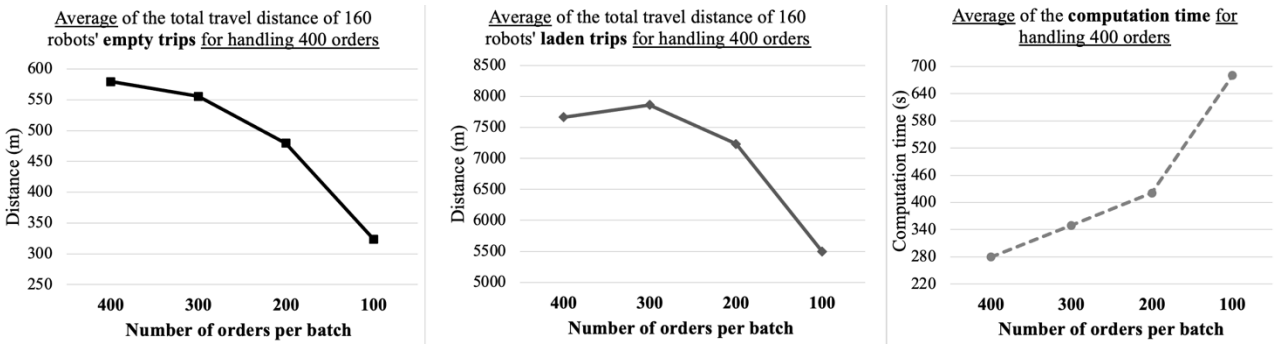


Figure 16: Performance of the multi-batch mode when handling 4800 orders

As mentioned, the three-stage based method is intuitive and also fast. To investigate the dynamic performance of the integrated-model based method relative to the three-stage method, more comparative experiments are performed in the above multi-batch context, in which the number of orders per batch is set as 400, 300, 200, or 100 orders, respectively, over a horizon containing 4800 orders in total. The results in Table 7 demonstrate the dynamic performance of the integrated-model based method, in which the integrated method is solved by the TRA. The batch size also has influence on the overall time performance of the integrated method; the larger the batch size is, the larger the saving on computation time for the three-stage based method.

Table 7: Comparison of computation time between the integrated-model based method and the three-stage based method in the multi-batch mode when handling 4800 orders

Number of orders per batch / Two decision methods	400	300	200	100
Integrated method's average computation time per 400 orders handled (T_1)	280 s	350 s	421 s	680 s
Three-stage method's average computation time per 400 orders handled (T_2)	207 s	264 s	352 s	564 s
(T_1—T_2)/ T_2	26%	25%	16%	17%

The above extended multi-batch model includes a new constraint limiting the number of stations that can be visited consecutively by a robot/pod in a single tour (i.e., the tour length), defined as parameter $\dot{t}$ (see Appendix I). Some experiments are conducted to investigate the influence of the tour length on the total waiting time of robots/pods at the stations, as this may reflect the congestion (or interference) in operations. The results

in Figure 17 imply that when the tour length limit decreases, the congestion in operations (reflected by the total waiting time) may initially not change significantly. However, for smaller values for the limit, the congestion may become severe, as the total waiting time increases substantially. This result can be explained, because at a high limit value, more pod scheduling solutions are available than with a low limit. However, from a practitioner's view, limiting each robot to visit fewer stations should lead to less potential congestion. The reason for the difference between the theoretical result and the practitioner' common sense lies in the fact that the length of tour (i.e., the number of stations visited by a robot) is a decision variable rather than a parameter in our problem. The robot may wait at more stations, but shorter periods in shorter queues. While a robot waiting at a few stations may have to wait in a longer queue.

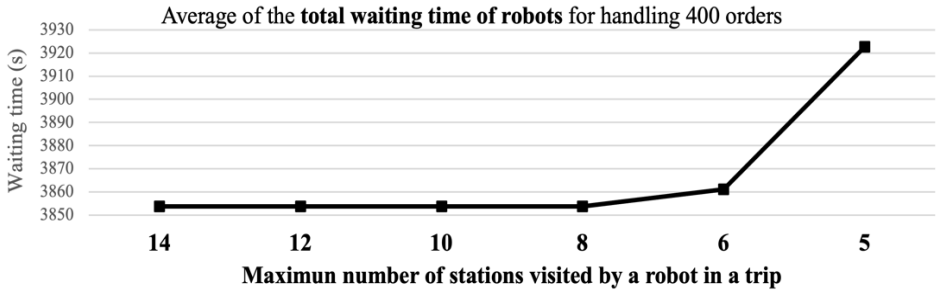


Figure 17: Influence of the tour length limit on the total waiting time of robots/pods at stations

The above experiments validate the applicability of the multi-batch model for handling a large number of orders (i.e., 4800 orders).

6.3 Integrating replenishing stations

The available inventory of an SKU influences the order picking process. As pod inventory decreases, it must be replenished timely (e.g., coming from an internal reserve storage location or a vendor). In an RMFS, workstations are multifunctional and can be rapidly converted from picking to replenishment or vice versa (Lamballais et al., 2022). Some RMFS-related studies explicitly consider pod replenishment (Roy et al., 2019; Jiang et al., 2020; Lamballais et al., 2020; Silva et al., 2020). This section extends the batch model to consider both picking and replenishment processes and formulates decision models for two strategies.

(1) *Separate*: a strategy in which the replenishment process is executed first (e.g., in the early morning), followed by the picking process. In this sequential replenishment strategy, the picking operations only begin after all replenishment work has been completed.

(2) *Concurrent*: a strategy in which picking and replenishment are executed simultaneously.

In both strategies, replenishment should be executed if a SKU's stock is below a threshold (see Appendix H, the parameter $\check{f}_{ps}$ is the threshold). The quantity of an SKU to be replenished is a decision variable. An SKU on a pod is replenished to its maximum level. The resulting models for the two strategies (separate or concurrent) are extensions of M_{Batch} and are denoted by M_{Spt} and M_{Con}, respectively. Appendix H includes details of the two models.

We compare the two pod replenishment strategies following Silva et al. (2020). They use the terms 'integrated planning' to denote the concurrent strategy and 'sequential planning' to denote the separate strategy. We use CPLEX to solve the models for a small-scale example with two pick stations and two replenishment

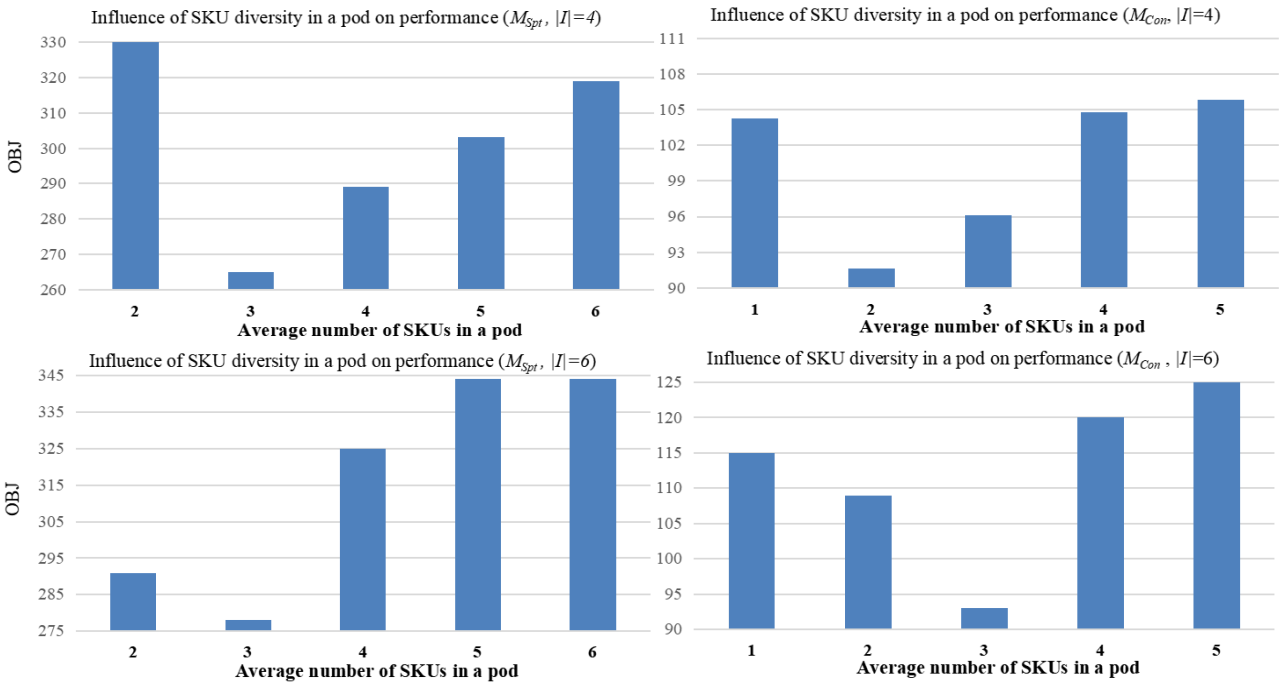
stations. The layout is based on Figure 6. The two pick stations are located along one long side, as in ISG 1 in Figure 6, and the two replenishment stations are located along the other long side of the warehouse.

The results in Table 8 show that the concurrent strategy performs much better than the separate strategy, with an average relative advantage of 54%. Roy et al. (2019) found a similar effect. They compared pooled (i.e., concurrent) versus dedicated (i.e., separate) robots for picking and replenishment. Their experimental results indicate that pooling can reduce overall pod waiting time at stations by up to 60%. The results in Table 8 are also in line with Jiang et al. (2020), who demonstrate that picking-replenishment synchronization (like the concurrent strategy) significantly outperforms an unbalanced strategy. They show that the performance gap increases when the problem scale grows. This can also be seen in the rightmost column in Table 8. The average gap percentage in large-scale instances is larger than in small-scale ones. These results demonstrate that the concurrent (or synchronization) picking and replenishment strategy is well-suited for RMFSs, especially in larger-scale applications.

Table 8: Comparison of separate and concurrent replenishment strategies

$\|I\|$	$\|S\|$	$\|P\|$	#	OBJ of M_{Spt} Z_{Spt}	Time (s)	OBJ of M_{Con} Z_{Con}	Time (s)	$\frac{Z_{Spt}-Z_{Con}}{Z_{Spt}}$
4	8	5	1	137.7	76	93.8	148	32%
			2	110.7	11	83.6	325	24%
			3	366.9	12	66.2	394	82%
			4	332.9	7	96.9	354	71%
			5	93.5	8	83.3	212	11%
			6	189.5	18	95.6	158	50%
6	8	5	1	271.9	9	72.5	123	73%
			2	268.5	10	135.1	1864	50%
			3	295.4	20	95.6	2652	68%
			4	180.8	110	76.0	1518	58%
			5	269.5	18	89.8	1874	67%
			6	329.3	14	109.8	1680	67%
						Average：		54%

Based on the extended models M_{Spt} and M_{Con} that include both picking and replenishment, we conduct some further experiments on the two small-scale problem instance groups to investigate the influence of the SKU diversity in a pod on performance. With a limited number of SKUs per pod, the pods do not need frequent replenishment, but more time is required to retrieve all the pods needed for a given set of orders. Figure 18 shows the result. When the average number of SKUs per pod increases, performance will first improve, reducing the makespan. However, when the SKU diversity in each pod continues to increase, the inventory level of each SKU in each pod decreases, leading to a higher replenishment frequency of each pod and lengthening the makespan. This phenomenon occurs in both replenishment strategies, as illustrated in Figures 18(a) and 18(b).



(a) Makespan with *separate* replenishment strategy (b) Makespan with *concurrent* replenishment strategy

Figure 18: Influence of SKU diversity per pod on makespan for two replenishment strategies

7. Conclusions

This paper presents novel, comprehensive MIP models to schedule all short-term activities in an RMFS to minimize order makespan. While several papers focus on partial aspects, such as product-to-pod or order-to-station assignment, these decisions are interrelated. However, these model types are notoriously difficult to solve, while they should be resolved quickly to be usable in practice. In line with current practice, we formulate an intuitive three-stage *sequential* decision framework and an *integrated* model with a fast, practical algorithm. We extend the model to include semi-batches (i.e., taking ongoing processes into account) and concurrent picking and replenishment operations. We develop a metaheuristic solution algorithm that can solve complex models with 14 stations, 400 orders, 300 SKUs, 160 pods, and 160 robots within four minutes. It is near-optimal for small instances. This time is sufficiently short to be combined e.g., in a rolling horizon approach, to solve the problem in real-life situations. This study also paves the way for the design of scheduling software for RMFS warehouse management systems.

We conduct extensive numerical experiments to validate the efficiency of the solution algorithm and the effectiveness of the decision models. We show that integrating decisions leads to significant benefits compared to a sequential approach and the heuristic scheduling rules described by Merschformann et al. (2019). Our experiments lead to the following managerial and design insights.

Managerial insights. Our experimental results show that it does not pay off to optimize isolated decisions such as assigning inventory to pods, orders to pods and workstations, robots to pods, pods to storage locations, sequencing orders, and sequencing station visits by the robots. Decisions are intertwined and should be integrated. Integration, even when done heuristically, is beneficial compared to sequential optimization of isolated decisions. This is also supported by Merschformann et al. (2019), who compare heuristic rules by simulation. We demonstrate that the SKU diversity per pod and over pods strongly impacts the total completion

time for handling order batches. It allows using a given pod to fill more orders (by routing pods to the different pick stations in sequence) and to select a pod closer to a pick station, decreasing robot travel time. However, increasing the number of SKUs per pod may lead to many concurrent replenishments, extending the makespan. We also found that the concurrent replenishment strategy outperforms the separate (i.e., sequential) replenishment strategy. This suggests it might be beneficial to dynamically adjust the number of workstations assigned to picking and replenishment tasks (as proposed by Lamballais et al., 2022). More research should focus on quantifying the benefits of such a strategy.

Design insights. Warehouse layout, such as the position of stations, can influence performance (e.g., Lamballais et al., 2017). Since the layout in an RMFS is much more flexible than in rack-based warehouses, these insights have managerial implications. Better performance can be achieved by locating pick stations along one of the long sides of the pod storage area. The number of pick stations and the order composition (the number of order lines and the sizes of SKUs in an order) influence performance.

Our work opens many avenues for further research. Faster solution algorithms are needed as they allow using the proposed methodology to even larger scale problems. We could investigate some problems with more dynamic design-related decisions, such as changing the number of pick and replenishment stations and the pod storage layout (length/width ratio and the number of cross aisles). Explorative studies could combine MIP formulations and queuing network methods, as queuing networks work well to solve strategic-level decision problems such as optimal layout or capacity design for RMFSs. Future studies could also analyze the combined effects of optimal decision-making and human factors in managing RFMSs.

Acknowledgement

Thanks are due to the referees and editors for their valuable comments that helped improve the quality of this paper. The research is supported by the National Natural Science Foundation of China (Grant numbers 72025103, 72361137001, 71831008, 72201229, 72361137006), and the Research Grants Council of the Hong Kong Special Administrative Region, China [Project Number 25211722].

References

Azadeh, K., R. de Koster, D. Roy (2019) Robotized and Automated Warehouse Systems: Review and Recent Developments. *Transportation Science* 53(4): 917–945.

Boysen, N., D. Briskorn, S. Emde (2017) Parts-to-picker based order processing in a rack-moving mobile robots environment. *European Journal of Operational Research* 262(216): 550–562.

Boysen, N., D. Briskorn, S. Fedtke, M. Schmickerath (2019a) Automated sortation conveyors: A survey from an operational research perspective. *European Journal of Operational Research* 276(31): 796–815.

Boysen, N., R. de Koster, F. Weidinger (2019b) Warehousing in the e-commerce era: A survey. *European Journal of Operational Research* 277(21): 396–411.

Boysen, N., S. Schwerdfeger, M. W. Ulmer (2023) Robotized sorting systems: Large-scale scheduling under real-time conditions with limited lookahead. *European Journal of Operational Research*. In Press.

Businesswire (2020) Alibaba Generates RMB498.2 Billion (US$74.1 Billion) in GMV During the 2020 11.11 Global Shopping Festival. Accessed Nov 12, 2020, https://www.businesswire.com/news/home/20201111005881/en/.

Dawande, M., M. Johar, S. Kumar, V. S. Mookerjee (2008) A comparison of pair versus solo programming under different objectives: An analytical approach. *Information Systems Research* 19(1): 71–92.

Garey, M. R., D. S. Johnson (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York.

Goeke, D., M. Schneider (2021) Modeling Single-Picker Routing Problems in Classical and Modern Warehouses. *INFORMS Journal on Computing* 33(2): 436–451.

Gzara, F., S. Elhedhli, U. Yildiz, G. Baloch (2020) Data-Driven Modeling and Optimization of the Order Consolidation Problem in E-Warehousing. *INFORMS Journal on Optimization* 2(4): 229–346.

Hou, R., R. de Koster, Y. Yu (2018) Service Investment for Online Retailers with Social Media——Does It Pay Off? *Transportation Research Part E: Logistics and Transportation Review* 118: 606–628.

Jiang, M., K. H. Leung, Z. Lyu, G. Q. Huang (2020) Picking-replenishment synchronization for robotic forward-reserve warehouses. *Transportation Research Part E: Logistics and Transportation Review* 144: 102138.

Lamballais, T., M. Merschformann, D. Roy, R. de Koster, K. Azadeh, L. Suhl (2022) Dynamic policies for resource allocation in a robotic mobile fulfillment system with time-varying demand. *European Journal of Operational Research* 300(3): 937–952.

Lamballais, T., D. Roy, R. de Koster (2017) Estimating performance in a Robotic Mobile Fulfillment system. *European Journal of Operational Research* 256(31): 976–990.

Lamballais, T., D. Roy, R. de Koster (2020) Inventory allocation in robotic mobile fulfillment systems. *IISE Transactions* 52(1): 1–17.

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

Miller, C. E., A. W. Tucker, R. A. Zemlin (1960). Integer programming formulations and traveling salesman problems. *Journal of Association for Computing Machinery* 7: 326–329.

Nigam, S., D. Roy, R. de Koster, I. Adan (2014) Analysis of class-based storage strategies for the mobile shelf-based order pick system. In: Smith, J., K. Ellis, R. de Koster, S. Lavender, B. Montreuil, M. Ogle (eds.), Progress in Material Handling Research 2014, *13th IMHRC Proceedings* (CICMHE, Charlotte, NC), 19.

Qin, H., J. Xiao, D. Ge, L. Xin, J. Gao, S. He, H. Hu, J. G. Carlsson (2022) JD.com: operations research algorithms drive intelligent warehouse robots to work. *INFORMS Journal on Applied Analytics* 52(1): 42–55.

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

Silva, A., L. C. Coelho, M. Darvish, J. Renaud (2020) Integrating storage location and order picking problems in warehouse planning. *Transportation Research Part E: Logistics and Transportation Review* 140: 102003.

Shi, Y. H. Yu, Y. Yu, X. Yue (2021) Analytics for IoT‐enabled human-robot hybrid sortation: an online optimization approach. *Production and Operations Management*. In Press.

Tseng, P., S. Yun (2009) A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming* 117(1-2): 387–423.

Valle, C. A., J. E. Beasley (2021) Order allocation, rack allocation and rack sequencing for pickers in a mobile rack environment. *Computers & Operations Research* 125: 105090.

Wang, Z., J.-B. Sheu, C.-P. Teo, G. Xue (2022) Robot Scheduling for Mobile-Rack Warehouses: Human–Robot Coordinated Order Picking Systems. *Production and Operations Management* 31(1): 98–116.

Wang, B., X. Yang, M. Qi (2022) Order and rack sequencing in a robotic mobile fulfillment system with multiple picking stations. *Flexible Services and Manufacturing Journal*. In Press.

Weidinger, F., N. Boysen (2018) Scattered Storage: How to Distribute Stock Keeping Units All Around a Mixed-Shelves Warehouse Storage. *Transportation Science* 52(6): 1412–1427.

Weidinger, F., N. Boysen, D. Briskorn (2018) Storage Assignment with Rack-Moving Mobile Robots in KIVA Warehouses. *Transportation Science* 52(6): 1479–1495.

Wurman, P. R., R. d'Andrea, M. Mountz (2008) Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine* 29(1): 9–19.

Xie, L., N. Thieme, R. Krenzler, H. Li (2021) Introducing split orders and optimizing operational policies in robotic mobile fulfillment systems. *European Journal of Operational Research* 288(1): 80–97.

Yang, X., G. Hua, L. Hu, T. C. E. Cheng, A. Huang (2021) Joint optimization of order sequencing and rack scheduling in the robotic mobile fulfilment system. *Computers & Operations Research* 135: 105467.

Yu, M., R. de Koster (2008) Performance approximation and design of pick-and-pass order picking systems. *IIE Transactions* 40(11): 1054–1069.

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

Yuan, R., S. C. Graves, T. Cezik (2019) Velocity-Based Storage Assignment in Semi-Automated Storage Systems. *Production and Operations Management* 28(2): 354–373.

Yun, S., P. Tseng, K. C. Toh (2011) A block coordinate gradient descent method for regularized convex separable optimization and covariance selection. *Mathematical Programming* 129(2): 331–355.

Zhang, J., S. Onal, S. Das (2020) The dynamic stocking location problem – Dispersing inventory in fulfillment warehouses with explosive storage. *International Journal of Production Economics* (224), 107550

Zhen, L., E. P. Chew, L. H. Lee (2011) An integrated model for berth template and yard template planning in transshipment hubs. *Transportation Science* 45(4): 483–504.

Zhen, L., H. Li (2022) A literature review of smart warehouse operations management. *Frontiers of Engineering Management* 9(1): 31–55.

Zou, B., Y. Gong, X. Xu, Z. Yuan (2017) Assignment rules in robotic mobile fulfillment systems for online retailers. *International Journal of Production Research* 55(20): 6175–6192.

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