



Order and rack sequencing in a robotic mobile fulfillment system with multiple picking stations

Bingqian WANG¹ · Xiuqing YANG² · Mingyao QI³ 

Accepted: 29 October 2021 / Published online: 17 January 2022

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2021

Abstract

The robotic mobile fulfillment system, where mobile robots carry racks to stationary stations for human pickers, is widely used in e-commerce warehouses. The operational efficiency of the system is largely affected by the organization of the order picking process. In this paper, we investigate an order picking tactic in a robotic mobile fulfillment system that allows racks moving between multiple picking stations in order to feed more orders during one visit and therefore save the overall throughput time. To evaluate this operational tactic's effectiveness, we propose a mathematical model to jointly optimize the order assignment, order sequencing, rack selection, and rack sequencing. A two-stage hybrid heuristic algorithm framework is then presented, including order assignment in the first stage and order and rack sequencing in the second stage. We conduct numerical experiments to validate the proposed algorithms' performance under different strategies and find out that the inter-station operation can significantly save order throughput time on the testing cases. We also investigate the effects of several factors, including the number of picking stations, their capacities, the stock keeping unit diversity, and queue length. Furthermore, a solid simulation is carried out to show the rationale of using rack moves as the objective rather than the completion time.

Keywords Logistics · Material handling · Order picking · Robotic · Heuristics · Simulation

1 Introduction

In order fulfillment warehouses, about 55% of the operating costs are due to the order picking process (De Koster et al. 2007; Jaghbeer et al. 2020). To improve operational efficiency and save costs, automated and robotic handling systems are

✉ Mingyao QI
qimy@sz.tsinghua.edu.cn

Extended author information available on the last page of the article

increasingly applied in warehouses (Azadeh et al. 2019). Among them, Robotic Mobile Fulfillment System (RMFS), pioneered by the Kiva system (Wurman et al. 2008), now rebranded as Amazon Robotics, has been applied in e-commerce distribution centers in recent years. Compared to traditional picker-to-parts warehouse systems, RMFS facilitates a parts-to-picker order picking mode. Robotic devices bring items from a storage area to a picking station, where human pickers complete the order picking and possible packing (Huang et al. 2015). Therefore, it can save the picker's travel time along aisles and can even double the warehouse productivity (Wurman et al. 2008). Meanwhile, compared to the miniload system, another automated parts-to-picker system facilitated by aisle-captive Storage/Retrieval machines (such as DEMATIC), RMFS has the virtue of incremental scalability, (ease of) parallel processing, and built-in redundancy (Bozer and Aldarondo 2018). In the industry, E-commerce companies, such as Amazon.com, JD.com, Tmall.com, and so on, have reported their RMFS applications. Meanwhile, following Amazon Robotics, many device and solution providers in this industry emerge, including Swisslog, Grey-Orange, Hitachi, and QuickTron, among others.

In practice, one RMFS is often equipped with multiple picking stations (PS), each one of them is in charge of processing a group of orders typically done by one picker (Merschformann et al. 2019). To avoid pickers waiting for the arrival of movable racks, a queue of racks with given length is often arranged to line in front of a picking station (PS) so that a finished rack can be replenished instantly. To illustrate, the layout of a warehouse with four PS is presented in Fig. 1 in which the length of the waiting queue is 3. The PS are located on the left side of the warehouse (called picking area), and the movable racks containing specific SKUs (stock keeping units) are placed on the right part (called the storage area). An SKU can be stored in multiple racks and a rack can also contain multiple types of SKUs. Once a set of orders arrives at the system and is to be retrieved, the first decision to be made is the order assignment determining which orders should be assigned to which PS. Subsequently, the processing sequence of the

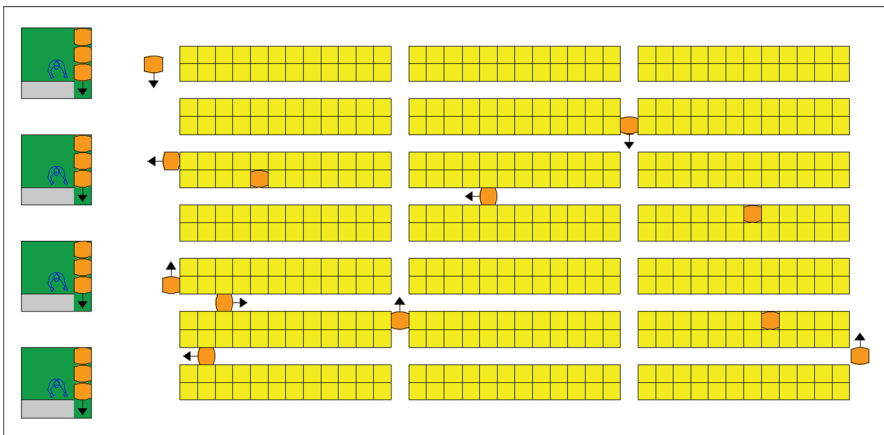


Fig. 1 A typical warehouse layout of an RMFS

orders in each PS should be specified. For each PS, processing a given batch of orders requires a set of racks which contains the necessary SKUs that the orders require. Since an SKU can be stored in multiple racks (for example, in a random way), racks that can satisfy the orders need to be selected, as well as the visiting sequence of these racks. As Boysen et al. (2017a) and Valle and Beasley (2021) indicate, both the order sequence and the rack sequence significantly affect the efficiency of the system. However, they only handled order picking issues with only one PS, and we will extend it to adapt to multiple-PS scenarios in this study.

There are a bunch of studies addressing order picking problems in traditional manual warehouses and just a few for parts-to-picker systems; however, as far as we know, existing research does not consider the scenario that a rack serves two or more PS during its single PS-visiting tour (hereafter referred as *inter-PS service*). Intuitively, this would be very helpful if the SKUs contained in a visiting rack can satisfy the in-process orders of multiple PS. Imagine that two adjacent PS are handling orders with similar contents, if a rack consecutively service these two PS, rather than visiting them separately, a rack trip from the storage area to the picking area would be saved. Actually, this operation mode has been practiced by Quicktron, a robotic fulfillment solution company headquartered in Shanghai, China that offers intelligent warehousing solutions (QuickTron 2020). With this idea in mind, we investigate the order processing in a parts-to-picker system with multiple PS, which involves multiple decisions, such as order assignment, order sequencing, rack selection, and rack sequencing. The total time of completing all given orders is usually used to measure the efficiency of a traditional order processing system. However, setting the completion time as the objective would dramatically increase the scale of decision variables (for example, variables to represent the exact rack moving routes), and would make the mathematical model computationally intractable. Moreover, it is also hard to measure some factors in RMFSs that affect the completion time, for example, the route conflicts that occur during the moving of racks. In this paper, we adopt a surrogate objective, which is the number of rack moves necessary to complete all orders (Boysen et al. 2017a; Valle and Beasley 2021). Furthermore, we reveal in this study by a simulation that the number of rack moves between PS and the storage area (excluding inter-PS rack moves) is linearly related to the order completion time.

The contributions of this paper are as follows: 1) Enlightened by the practice of the industry, this work for the first time considers the inter-PS operation during the order picking process in an RMFS; 2) Enabling such inter-PS operations, we propose an integrated mixed-integer linear programming model to jointly optimize the order assignment, order sequencing, rack selection, and rack sequencing of the RMFS, which can be used as a benchmark for the evaluation of heuristic algorithms; and 3) Based on the proposed two-stage hybrid algorithm, we observe that enabling inter-PS operation can substantially improve order picking efficiency.

The remainder of this paper is organized as follows. Section 2 reviews the related literature. Section 3 describes the problem in detail and formulates an integer programming model. Section 4 proposes a two-stage algorithm framework and show the implementation of each stage. Computational experiments and the results are

presented in Sect. 5. Then, Section 6 elaborates the simulation work to prove the rationale of using rack moves as the objective rather than the completion time. Finally, Section 7 draws the conclusion and points out future research suggestions.

2 Literature review

Order picking and the associated warehouse design problems have long attracted intensive research interest for decades. Gu et al. (2007) and De Koster et al. (2007) provided comprehensive reviews on layout design, storage assignment, order picking and routing. In the up-to-date review works, Azadeh et al. (2019); Jaghbeer et al. (2020) focus on the topic of robotics in order picking. This work falls in the category of “human picker” and “parts-to-picker” in Jaghbeer et al. (2020), and the category of Robotic Mobile Fulfillment System in Azadeh et al. (2019). As one of the directions for future research pointed out in Azadeh et al. (2019), an integrated model is also pursued in this study; that is, we propose a model that jointly optimize the order assignment (to picker), order sequencing, rack selection, and sequencing. The objective is to optimize the overall performance over multiple PS.

The related research on order picking problems are mostly dedicated for picker-to-parts systems, for example, see Öncan (2015), Menéndez et al. (2016), Scholz et al. (2017), among others. We also refer to a review-based study (Grosse et al. 2015) which investigates the human factor aspects involved in order picking processes, particularly for manual systems. However, we make our focus on the order picking problems in Robotic Mobile Fulfillment Systems. We summarize related literature in Table 1, and a survey paper by Boysen et al. (2019) is also consulted.

As shown in Table 1, most studies address only one or two isolated steps of the whole order picking process, which we summarize as order assignment, order

Table 1 Summarization of relevant literature

Reference	Order assignment	Robot assignment	Rack selection	Rack sequencing	Order sequencing	Solution methods
Zou et al. (2017)		✓				Heuristic
Li et al. (2017)			✓			Heuristic
Boysen et al. (2017a)			✓	✓	✓	DP
Boysen et al. (2017b)					✓	Heuristic
Boysen et al. (2018)					✓	DP
Xi et al. (2018)			✓			Heuristic
Zhang et al. (2019)		✓				Heuristic
Füßler and Boysen (2019)				✓	✓	Heuristic
Merschformann et al. (2019)	✓		✓			Rules-based
Valle and Beasley (2021)	✓		✓	✓	✓	Heuristic
This paper	✓		✓	✓	✓	Heuristic

sequencing, rack selection, rack sequencing, and robot assignment. Zou et al. (2017) study the rules of assigning PS to robots according to PS's handling speed. A neighborhood search algorithm is designed to find a near-optimal assignment rule, and semi-open queueing networks are built to estimate the system performance. This study considers only single-line orders. Xi et al. (2018) focus on the storage assignment and order batching problem in a Kiva mobile fulfillment system based on product similarity, aiming to minimize the number of visits of racks. They also determine the assignment of racks to a given batch. However, their study doesn't consider the sequence that racks visit a PS. Fäßler and Boysen (2019) treat the processing sequence of orders in a single picking station to reduce the number of storage bins transferred from the storage system to fulfill orders. This problem is relatively less complicated than ours since each storage bin contains a single SKU. Zhang et al. (2019) tackle The robot allocation problem under a determined rack sequence in each PS. They propose a model based on the resource-constrained project scheduling problem with transfer times to minimize the system's makespan and use a designated genetic algorithm to solve the problem. Li et al. (2017) determine the optimal rack selection for a given batch of orders under parts-to-picker mode. Aiming to minimize the total time of moving the selected racks to finish orders, they propose a three-stage hybrid heuristic algorithm. However, the racks' arrival sequence has not been optimized, and only one batch of orders is considered. Similarly, Boysen et al. (2018) focus on the release sequence of bins from the automated storage/retrieval system (AS/RS) to minimize the spread of orders so that orders can be quickly assembled at their packing stations. Boysen et al. (2017b) consider the order sequencing in mobile rack warehouses, in which there are only a few aisles open for rack moves at a time. Therefore, aisle relocation is also subject to optimization. Finally, Merschformann et al. (2019) evaluate multiple decision rules for several problems in RMFSs by simulation: the order assignment, rack selection, and rack storage assignment. The pick order assignment rules are based on the slot of the just-completed order at a station, which means orders are assigned to a PS from the backlog and only one is assigned at a time, except for the rare case in which multiple orders are completed at the same time. The simulation approach is rule-based and myopic in nature.

We highlight two pieces of research which we think are mostly close to ours. Boysen et al. (2017a) for the first time point out that both the order processing sequence and the rack arrival sequence at the PS notably affect the order picking efficiency in a robotic parts-to-picker system. They provide a mathematical formulation of the joint problem and propose a decomposition-based solution approach that incorporates two dynamic programming (DP) algorithms. It is worth noting that, though their study can theoretically find an optimal solution, the computational time to get the optimal solution can easily go beyond the limit that people can wait. This is also the reason why they compromise to use a beam search heuristic. Valle and Beasley (2021) extend the work of Boysen et al. (2017a) by additionally considering the order and rack allocation to PS (or, pickers as in the paper). The (first-stage) order and rack allocation solution is then submitted to a (second-stage) rack sequencing problem for each picker where the order sequence is also determined. This study formulates and solves the two subproblems separately: for the order and rack allocation

problem, two heuristics are proposed; while for the rack sequencing problem, the proposed formulation explicitly considers the rack inventory positions which are ignored in Boysen et al. (2017a), and the CPLEX solver is used to generate a feasible (not optimal) solution. However, both the above studies don't assume that a rack can visit two or more PS before it returns to the storage area. Therefore, our research aims to investigate how this consideration will benefit operational efficiency.

Beyond the problems described above, there are a few studies optimizing other procedures in a parts-to-picker system. Weidinger et al. (2018) consider the problem of assigning racks to storage positions when a rack returns from a PS to the storage area. The objective of the problem is to minimize the total travel distance of the mobile robots to complete all the tasks. Similarly, Yuan et al. (2019) also focus on the storage assignment problem in a parts-to-picker system where the velocity-based storage policies are analyzed. Using multi-class closed queueing network models, Roy et al. (2019) analyze both order picking and replenishment processes in a mobile fulfillment system.

To summarize, various studies have been devoted to one or several order picking subproblems in both traditional picker-to-parts systems and newly-emerging parts-to-picker systems. However, to the best of our knowledge, there is rarely any study addressing the optimization in an RMFS with multiple PS, which has been practiced. To bridge this gap, we explicitly consider inter-PS service aiming to save processing time.

3 Problem description

In a robotic mobile fulfillment system, we consider the processing of a set of given orders in multiple PS. First of all, customer orders are assigned to different PS based on some strategies (introduced later). Each PS has a limit on the number of orders that are assigned to it and can only process a certain number of orders in parallel, denoted by w and C , respectively. At a PS, the picking sequence for each batch of orders needs to be decided, besides, the sequence of racks arriving at each PS is also to be decided. When an order is completely fulfilled at a PS, it will be replaced by another new order immediately. A movable rack, which is carried by a robot from the storage area, can visit multiple PS if required. Hence, two processing sequences are necessary when picking orders at each PS, which are the order sequence and the rack sequence, both are crucial to the system performance as shown by the following example. Throughout this paper, a *round rack move* refers to the process of carrying a rack from the storage area to the picking area and then returning the rack to the storage area. We differentiate *inter-PS rack move* as moving a rack from one PS to another during an inter-PS service. Note that a *round rack move* may or may not contain inter-PS rack moves depending on the picking schedule. Hereafter, *round rack move* is referred to as *rack move* for simplicity unless otherwise specified.

3.1 Assumptions and an example

In order to support our research, the following assumptions are made.

- Assume that the shared storage policy (Bartholdi and Hackman 2008) is adopted, that is, an SKU is not completely stored on a single rack, but rather on multiple racks. According to Lamballais et al. (2019), this would incur less order completion time in a parts-to-picker system.
- Presuppose that there are enough units for each SKU on the rack. Based on this assumption, we can just focus on whether a rack contains an SKU, while the number of units is not critical (Boysen et al. 2017a). This assumption is particularly rational when we consider the e-commerce warehouses where the number of units for each SKU in an order is usually small, and an appropriate replenishment strategy can also reduce the probability of out of stock.
- During a rack visit at a PS, the order that has been fulfilled by this rack can be replaced by a new order immediately, and SKUs required by the new order can be retrieved from the current rack if they are present. A rack leaves the PS only when no SKUs on current picking orders are available on the rack.
- We assume the length of a time slot in different PS is the same, at least statistically. The violation of this assumption may incur asynchronism among PS and postpone the finish time of all the orders. We argue that this can be adjusted on a real-time scheme based on the solution deduced from our proposed model. To validate the effectiveness of this assumption, a simulation is also conducted as elaborated in Sect. 6.
- We assume that there is a buffer in front of each PS so that any pod that arrives earlier than scheduled can stay waiting.
- We focus on a warehouse layout where the PS are located on one side of the area. Thus, the transport distance between the PS is shorter than that between the storage area and PS in general.

Example Consider $S = 2$ PS and $N = 8$ customer orders in total. There are 4 orders to be picked at each station with the capacity of $C = 3$. We have $K = 4$ SKUs known as $\{a, b, c, d\}$. The set of orders for PS S_1 contains SKUs as follows: $O_{11} = \{a, b, c\}$, $O_{12} = \{c, d\}$, $O_{13} = \{b, c, d\}$, $O_{14} = \{a, b, c, d\}$, and orders for PS S_2 contains the following SKUs: $O_{21} = \{a, c\}$, $O_{22} = \{a, b\}$, $O_{23} = \{a, c, d\}$, $O_{24} = \{b, c, d\}$. In addition, we define $M = 3$ racks: $R_1 = \{a, c\}$, $R_2 = \{b, d\}$, $R_3 = \{c, d\}$. Two alternative solutions are proposed to process the orders in Fig. 2. In solution 1, the order sequence and rack arrival sequence at PS S_1 are $\langle O_{12}, O_{13}, O_{14}, O_{11} \rangle$ and $\langle R_3, R_1, R_2 \rangle$ respectively, and $\langle O_{21}, O_{22}, O_{23}, O_{24} \rangle$ and $\langle R_1, R_2 \rangle$ respectively at PS S_2 . In solution 2, the order sequence and rack arrival sequence at PS S_1 are $\langle O_{11}, O_{12}, O_{13}, O_{14} \rangle$ and $\langle R_1, R_3, R_2, R_1 \rangle$ respectively, and $\langle O_{22}, O_{24}, O_{21}, O_{23} \rangle$ and $\langle R_2, R_1, R_3 \rangle$ respectively at PS S_2 .

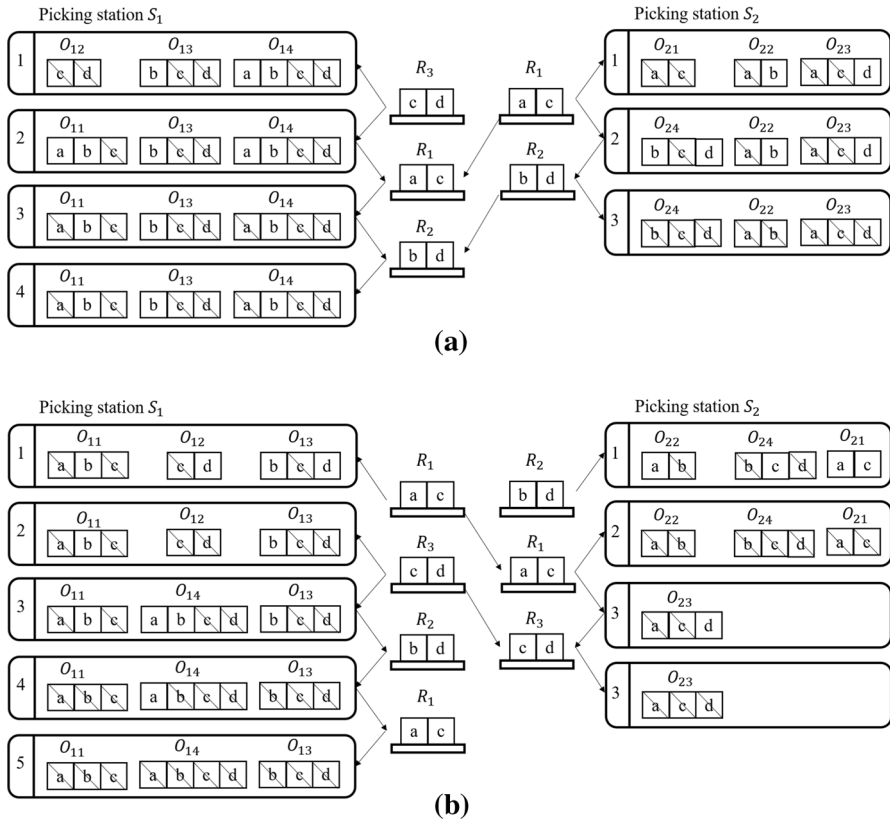


Fig. 2 An example of two order picking solutions with different order and rack sequences

As we can see from Fig. 2, five rack visits are required to complete all customer orders with solution 1 (R_3 , R_1 and R_2 visit S_1 , and R_1 and R_2 visit S_2), while seven visits are needed in solution 2 (R_1 , R_3 , R_2 and R_1 visit S_1 , and R_2 , R_1 and R_3 visit S_2). Adopted from Boysen et al. (2017a), the concept of time slot is used to track the process of the order picking and rack moving. In a time slot, for each PS, there is a certain number of orders in picking and only one rack is visiting. For each PS, when one or more orders are fulfilled during a rack visit, or, when there is a rack change, the time slot moves to the following. Therefore, 4 slots involve in solution 1 and 5 in solution 2. Clearly, if inter-PS service is not allowed, then in solution 1, the number of rack moves is 5 (3 for S_1 and 2 for S_2) and in solution 2, is 7 (4 for S_1 and 3 for S_2). However, if inter-PS service is allowed and we suppose a rack can be handled immediately when it is transferred to another PS (that is, the queue length is 1), then both solution 1 and 2 contain two inter-PS rack moves. As a result, the number of rack moves becomes 3 for solution 1 and 5 for solution 2. Considering that travel distance between PS

Table 2 Constant parameters

Symbol	Explanation
N	Number of customer orders to be retrieved
M	Number of racks
S	Number of PS
K	Number of SKU types
w	Maximum number of orders assigned to each PS
C	Capacity of each PS
Q	Length of queue at each PS
T	Maximum number of time slots
$Oset_{ik}$	1, if order i contains SKU k , $\forall i = 1, \dots, N, \forall k = 1, \dots, K$; 0 otherwise
$Rset_{jk}$	1, if rack j contains SKU k , $\forall j = 1, \dots, M, \forall k = 1, \dots, K$; 0 otherwise

Table 3 Decision variables

Decision variables	Explanation
y_{is}	Binary variables: 1, if order i is assigned to PS s ; 0 otherwise
x_{jst}	Binary variables: 1, if rack j is visiting PS s in slot t ; 0 otherwise
o_{ist}	Binary variables: 1, if order i is processed at PS s in slot t ; 0 otherwise
$z_{jss't}$	Binary variables: 1, if rack j is moving from PS s to PS s' in slot t ; 0 otherwise
L_{kist}	Binary variables: 1, if SKU k is delivered for order i at PS s in slot t ; 0 otherwise
r_{st}	Binary variables: 1, if the racks visiting at PS s in $t - 1$ and t differ; 0 otherwise

is much shorter than that from the storage area to PS, the total travel distance or time can be remarkably saved by allowing inter-PS service. This small example also reflects that order sequence and rack sequence have a great influence on order picking performance.

3.2 Mathematical model

Based on the above setting and assumptions, we build an integer programming model for this problem which consists of the objective function (1) and constraints (2) to (20). Before explaining the model, Table 2 shows the constant parameters and Table 3 shows the decision variables applied in the model.

$$\text{Minimize } \sum_{t=1}^T \sum_{s=1}^S r_{st} - \sum_{t=2}^{T-Q} \sum_{j=1}^M \sum_{s=1}^S \sum_{s'=1, s' \neq s}^S z_{jss'(t+Q)} \tag{1}$$

$$\text{s.t. } \sum_{s=1}^S y_{is} = 1, \forall i = 1, \dots, N, \tag{2}$$

$$\sum_{i=1}^N y_{is} \leq w, \forall s = 1, \dots, S, \tag{3}$$

$$\sum_{j=1}^M x_{jst} \leq 1, \forall s = 1, \dots, S, \forall t = 1, \dots, T, \tag{4}$$

$$\sum_{s=1}^S x_{jst} \leq 1, \forall j = 1, \dots, M, \forall t = 1, \dots, T, \tag{5}$$

$$1 - (x_{jst}x_{js't'} - |\sum_{t_1=1}^t r_{st_1} - \sum_{t_2=1}^{t'} r_{s't_2} - Q|) - M(1 - z_{js'st}) \leq 0, \\ \forall j = 1, \dots, M, \forall s = 1, \dots, S, \forall s' = 1, \dots, S, s' \neq s, \forall t = 1, \dots, T, \forall t' = 1, \dots, T, t' \neq t, \tag{6}$$

$$Q - |\sum_{t_1=1}^t r_{st_1} - \sum_{t_2=1}^{t'} r_{s't_2}| - M(1 - x_{jst}x_{js't'}) \leq 0, \\ \forall j = 1, \dots, M, \forall s = 1, \dots, S, \forall s' = 1, \dots, S, s' \neq s, \forall t = 1, \dots, T, \forall t' = 1, \dots, T, t' \neq t, \tag{7}$$

$$\sum_{i=1}^N o_{ist} \leq C, \forall s = 1, \dots, S, \forall t = 1, \dots, T, \tag{8}$$

$$\sum_{s=1}^S o_{ist} \leq 1, \forall i = 1, \dots, N, \forall t = 1, \dots, T, \tag{9}$$

$$o_{ist} + o_{is(t+d)} \leq 1 + o_{is(t+1)}, \forall i = 1, \dots, N, \\ \forall s = 1, \dots, S, \forall d = 2, \dots, T - t, \forall t = 1, \dots, T - 1, \tag{10}$$

$$y_{is} - \sum_{t=1}^T o_{ist} \leq 0, \forall i = 1, \dots, N, \forall s = 1, \dots, S, \tag{11}$$

$$\sum_{t=1}^T o_{ist} - My_{is} \leq 0, \forall i = 1, \dots, N, \forall s = 1, \dots, S, \tag{12}$$

$$\sum_{t=1}^T L_{kist} \geq Oset_{ik}y_{is}, \forall i = 1, \dots, N, \forall k = 1, \dots, K, \forall s = 1, \dots, S, \tag{13}$$

$$2L_{kist} \leq o_{ist} Oset_{ik} + \sum_{j=1}^M x_{jst} Rset_{jk}, \forall i = 1, \dots, N, \quad (14)$$

$$\forall k = 1, \dots, K, \forall s = 1, \dots, S, \forall t = 1, \dots, T,$$

$$r_{st} = \frac{1}{2} \sum_{j=1}^M |x_{jst} - x_{js(t-1)}|, \forall s = 1, \dots, S, \forall t = 2, \dots, T, \quad (15)$$

$$r_{s1} = 1, \forall s = 1, \dots, S, \quad (16)$$

$$r_{st} \geq 0, \forall s = 1, \dots, S, \forall t = 1, \dots, T, \quad (17)$$

$$y_{is} \in \{0, 1\}, \forall i = 1, \dots, N, \forall s = 1, \dots, S, \quad (18)$$

$$x_{jst} \in \{0, 1\}, \forall j = 1, \dots, M, \forall s = 1, \dots, S, \forall t = 1, \dots, T, \quad (19)$$

$$\begin{aligned} z_{jss't} &\in \{0, 1\}, \forall j = 1, \dots, M, \forall s = 1, \dots, S, \\ &\forall s' = 1, \dots, S, s' \neq s, \forall t = 2, \dots, T, \end{aligned} \quad (20)$$

$$L_{kist} \in \{0, 1\}, \forall i = 1, \dots, N, \forall k = 1, \dots, K, \forall s = 1, \dots, S, \forall t = 1, \dots, T, \quad (21)$$

$$o_{ist} \in \{0, 1\}, \forall i = 1, \dots, N, \forall s = 1, \dots, S, \forall t = 1, \dots, T. \quad (22)$$

Objective (1) minimizes the number of rack moves between the storage area and PS for all orders. The total number of rack changes among all PS is presented in the first term, and the second term denotes the number of rack moves between PS. As we will reveal in Sect. 6, the inter-PS rack move has very limited effect on the completion time. Constraints (2) ensure each order is only assigned to exactly one PS. Constraints (3) restrict the number of orders assigned to a PS that cannot exceed w . Constraints (4) and (5) assure for one slot that no more than one rack is visiting at a PS and that a rack is allowed to visit at most one PS.

Constraints (6) state the relationship between variables x and z when a rack moves from one PS to another. They state that if a rack consecutively visits two PS, then its positions in the rack sequences of the two PS have a difference of Q . If the difference is greater than Q , we believe that the rack should return to the storage area rather than waiting for another PS. Note that Constraints (6) are nonlinear but can be directly processed by solvers like CPLEX due to the binary natures.

Constraints (7) assure one rack cannot visit another PS within Q slots to avoid the rack sharing conflicts, since one rack is not allowed to cut in the line even though the rack is just released from the previous PS. Note that the second term indicates the difference of positions when a rack j visits two racks s and s' .

Constraints (8) limit the capacity of PS, that is, the maximum number of orders processed at one PS in a slot. Constraints (9) and (10) ensure each order can only be processed at one PS and each order is satisfied during a set of consecutive slots. Constraints (11) and (12) state the relationship between order assignment and picking, that is, the PS processes an order that is assigned to it, and the order must be fulfilled within T . Constraints (13) assure that each SKU k required by order i assigned to station s must be delivered. Constraints (14) indicates that if an SKU is to be retrieved to satisfy an order, then that SKU must be contained by the visiting rack. Finally, Equations (15) and (16) measure the number of rack changes at each PS in a slot. Constraints (17-22) define the decision variables of the problem.

The parts-to-picker based order picking problem has been proved an NP-hard problem, even for one PS (Boysen et al. 2017a). In addition, the problem in this paper considers extra decisions like order assignment to PS and inter-PS rack moves. To solve the problem, we design a two-stage hybrid heuristic algorithm as elaborated in the following section.

4 The two-stage heuristic algorithm

In this section, we propose a two-stage hybrid heuristic algorithm to solve the problem. In the first stage of the algorithm, an order assignment problem is solved to assign orders to PS (Sect. 4.1). Subsequently, in the second stage, for each PS, orders and racks are sequenced with considering the inter-PS service (Sect. 4.2). We realize that this two-stage approach is essentially sequential and therefore no guarantee of global optimal; however, since the number of orders is often large, if we otherwise allow perturbation of orders from different PS, it would incur much computational burden. We also note that the main purpose of this study is to identify the benefits of enabling inter-PS service, therefore, we just feed the results of the first stage to the second and then observe the performance improvement comparing to individual-PS mode.

4.1 Order assignment

As we consider multiple PS in the order picking system, given a set of customer orders for processing, the first decision is to group the orders and assign them to different PS. We investigate two strategies of order assignment, i.e., Random strategy and Order Similarity strategy.

1. Random strategy

According to this simple rule, the customer orders are assigned to PS randomly. Note that orders are evenly assigned to PS so that the workload of each PS is balanced.

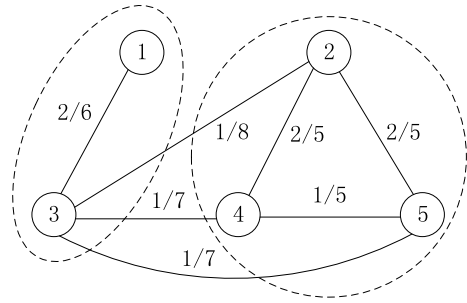
2. Order Similarity strategy

We define the similarity of two orders as the proportion of the number of common SKUs to the number of all SKUs that the two orders contain. According to the Order Similarity strategy, orders sharing more common SKUs tend to be assigned to the same PS. In this way, each rack visit will potentially fulfill more orders and therefore the rack moves needed to complete all orders will be reduced. Denote vector $O_i = (Oset_{i1}, Oset_{i2}, \dots, Oset_{iK})^T$, then the order similarity $\gamma_{ii'}$ of order i and order i' can be measured by Tanimoto coefficient (Tanimoto 1957):

$$\gamma_{ii'} = \frac{O_i^T O_{i'}}{O_i^T O_i + O_{i'}^T O_{i'} - O_i^T O_{i'}}. \quad (23)$$

In fact, the numerator denotes the number of common item types of the two orders, and the denominator is the total number of item types in both orders. With the definition of order similarity, the order assignment subproblem is to find the maximum

Fig. 3 An example to the order assignment algorithm according to similarity



order similarity assignment of orders to PS, so that each order is exactly assigned to one PS, and the maximum number of orders each PS can handle is respected. This problem falls into the category of a clustering problem. We implement a heuristic similar to Li and Li (2015), the procedure of which can be summarized as:

- Step 1 Calculate the similarity between any two orders based on Equation (23), and then use the results as the weights of edges to construct a weighted graph with N vertices, each standing for an order.
- Step 2 Sort by ascending the edge weights (similarity between orders).
- Step 3 Delete the edge with the smallest edge weight in the weighted graph.
- Step 4 Use the depth-first search algorithm to calculate the number of connected branches of the graph after an edge is deleted. If the number of connected branches is equal to the number of picking stations, go to Step 5; otherwise, go to Step 3.
- Step 5 Adjust the branches so that the number of vertices in each connected branch does not exceed w to ensure the balance of the workload between the PSs. For the connected branches with more than w vertices, select the vertex connected with the smallest number of edges, and delete the edges connected with it. Then the average weight between the vertex and the vertices in each connected branch whose number of vertices is less than w is calculated. Find a connected branch corresponding to the maximum average weight, and add the edge between this vertex and the connected branch.
- Step 6 Assign the vertices contained in each connected branch, that is, the orders, to each picking station.
- Step 7 Output the orders assigned to each picking station.

Example Consider 2 PS and 5 orders which are $O_1 = \{a, b, c\}$, $O_2 = \{d, e, f, g\}$, $O_3 = \{a, b, e, h, i\}$, $O_4 = \{d, f, h\}$, $O_5 = \{f, g, i\}$. Following the above steps, we can assign the 5 orders into 2 groups, one for orders 1 and 3, the other for orders 2, 4 and 5 (let $w = 3$), as shown in Fig. 3. The number in the circle is the order index, and the text along arc is the similarity of two orders.

4.2 Order and rack sequencing

In this second stage, we are going to find an optimal or “good” order sequence as well as a rack sequence for each PS. Unfortunately, the two sequences have mutual affects and can’t be solved separately. To address this problem, following the mechanism from Boysen et al. (2017a), we decompose the the order and rack sequencing problems into two interlaced subproblems: the order sequencing problem (OS) for a given rack sequence, and the rack sequencing problem (RS) for a given order sequence. Therefore, the order and rack sequences can be obtained from either perspective: explore the space of order sequences embedded with RS-solving steps, or vice versa. The two approaches are introduced in Sects. 4.2.2 and 4.2.3, respectively, and a comparison is conducted in Sect. 5.

When we consider a system with inter-PS service, we are facing the risk of rack conflicts between multiple PS. For example, a rack may be needed by two PS at the same time. It is worth noting that in practice, there are multiple racks queueing at each PS, since a lot of time would be wasted waiting for arrival of a new rack if not. Considering the queue length further complicates this problem as the possibility of rack conflict rises accordingly. Therefore, before getting into the details of the OS and RS algorithms, we introduce the rack conflict and its resolution in Sect. 4.2.1.

4.2.1 Rack conflict and its resolution

Suppose rack r is both in two sequences of PS a and PS b . Given rack sequence S_a at PS a and rack sequence S_b at PS b , with rack r at position p_a^r in S_a and position p_b^r in S_b , $p_a^r \leq p_b^r$, the distance between two positions associated with the same rack r , denoted as $D_{ab}^r = p_b^r - p_a^r$, has three conditions:

- $D_{ab}^r = Q$. Under this condition, the rack can exactly join the tail of the queue of PS b after servicing PS a , therefore, an inter-PS service will be taken.
- $D_{ab}^r > Q$. The rack will return to the storage area without servicing PS b since it otherwise must wait for other racks that in ahead of it according to S_b .
- $D_{ab}^r < Q$. This is the rack sharing conflict that we should avoid as the rack cannot cut in the line of PS b .

Therefore, the main idea of tackling the rack sharing conflicts is to ensure that for each rack that needed by multiple PS, its positions in different sequences must be not within distance of Q .

To resolve rack conflicts, we generate rack sequences in a sequential way for each PS, and each time a new rack sequence is built, it is checked with all previously generated rack sequences. If any conflict is detected, the following conflict elimination steps are taken on the current rack sequence only:

Step 1 For each conflict position of the sequence, swap it with another position so that conflicts with any previous sequences can be eliminated.

Step 2 If there is no such position to swap, let this conflict position be empty which means no rack (or a virtual rack) will come at this time slot, and then append the associated rack at the end of the sequence. Add more virtual racks before the rack if necessary.

Step 3 Repeat steps 1 and 2 on all other conflict positions.

4.2.2 The order sequencing algorithms

First, we introduce a greedy algorithm, called OS-Greedy, to deduce an order sequence under a known rack sequence, then we propose a Variable Neighborhood Search (VNS) based metaheuristic algorithm, named OS-VNS, to generate the order sequence with the rack sequence as a byproduct.

1. OS-Greedy

The OS-Greedy algorithm is to generate appropriate order picking sequences for each PS one by one when their rack sequences are fixed. As a pre-processing step, each rack sequence is checked with accomodated rack sequences and is adjusted if there is any rack conflict among them (see Sect. 4.2.1).

The basic idea of the greedy algorithm is to gradually add in unprocessed orders such that the current visiting rack can provide as many SKUs as possible. If some of the in-process orders are completed, new orders will be added in to replace the finished ones. When no more items can be retrieved from the visiting rack, the next rack in the given sequence comes. This process repeats until all orders are fulfilled. Specifically, given the current visiting rack r , the selection of an order to be added in can be expressed as the following order selection problem

$$\underset{i=1,\dots,N}{\operatorname{argmax}} \sum_{k=1}^K \left(\begin{cases} 1, & Oset_{ik} > 0 \wedge Rset_{rk} > 0 \\ 0, & \text{otherwise} \end{cases} \right). \tag{24}$$

For the convenience of calculation, the number of common SKUs shared by an order and a rack can be reflected by the difference between them. In other words, problem (24) can be recast as

$$\underset{i=1,\dots,N}{\operatorname{argmin}} d_{ir} \tag{25}$$

where

$$d_{ir} = \sum_{k=1,\dots,K, Oset_{ik}=1} (1 - Rset_{rk}). \tag{26}$$

d_{ir} measures the number of different SKUs which are contained in order i while not in rack r . The pseudocode of the OS-Greedy algorithm is shown in Algorithm 1, in which *RemainingOrderSet* is the set of orders not completed and not in picking, *PickingOrderSet* is the set of in-process orders, “DiffMatrix(i , *TimeSlot*)” represents

the matrix of order $i = 1, \dots, N$ and rack $TimeSlot = 1, \dots, M$. Note that each time slot represents the incumbent working rack.

Algorithm 1: OS-Greedy

Input: *RackSequence*
Output: *OrderSequence*

```

1 Resolve rack conflicts in RackSequence if necessary;
2 for  $b$  in all PS do
3    $TimeSlot = 1$ ;
4   Initiate RemainingOrderSet as the set of orders in PS  $b$ ;
5   PickingOrderSet =  $\phi$ ;
6   OrderSequence( $b$ ) =  $\phi$ ;
7   while the number of completed orders  $< N_b$  do
8     for  $i \in RemainingOrderSet$  do
9       Calculate DiffMatrix( $i, TimeSlot$ );
10    end
11    Rank DiffMatrix( $\cdot, TimeSlot$ ) in ascending order;
12    while  $|PickingOrderSet| < C$  do
13      Extract order(s) according to DiffMatrix( $\cdot, TimeSlot$ ) and add into
14      PickingOrderSet ;
15      Remove the order(s) from RemainingOrderSet;
16      Pick items from the current rack;
17      Remove finished SKUs from their orders;
18      if any picking order is completed then
19        Add completed orders into OrderSequence( $b$ );
20        Remove completed orders from PickingOrderSet;
21      end
22    end
23     $TimeSlot = TimeSlot + 1$ ;
24  end

```

2. OS-VNS

Variable Neighborhood Search (VNS) (Mladenović and Hansen 1997) as a metaheuristic algorithm has been widely used to solve combinatorial optimization problems. VNS can systematically explore the solution space by changing the way of generating neighborhood solutions (called shaking), and making efforts to further improve the current neighborhood (called local search). If the shaking and local search update the best-known solution, the current neighborhood operators will be retained; otherwise, the next neighborhood operator is recruited. If the current operator is the last one, then we activate the first operator again; so on so forth. The more effective the neighborhood operator is, the more frequent it is applied. As reviewed above, the VNS algorithm framework has also been used for solving order picking problems by a few researchers, see, for example, Menéndez et al. (2016) and Scholz et al. (2017).

Fig. 4 An example of the 2-Exchange operator

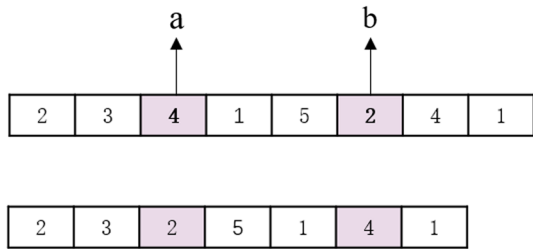
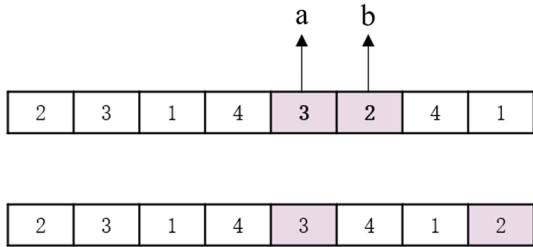


Fig. 5 An example of the Segment-Relocate operator



The pseudocode of the proposed OS-VNS algorithm is presented in Algorithm 2 where $\text{Shake}(rseq, k)$ means generating a neighborhood of rack sequence $rseq$ with k^{th} operator (elaborated later), and $\text{LS}(rseq', k_1)$ means taking local search on $rseq'$ with operator k_1 . The local search consists of taking $Iteration_{LS}$ times of

neighborhood operator k_1 on $rseq'$ and selecting the optimal one. Note that in our implementation of the local search, we use the first operator k_1 only due to its good performance in the experiments. The total number of neighborhood operators is denoted as k_{max} , and the stopping criterion is due to the maximum number of iterations $Iteration_{max}$.

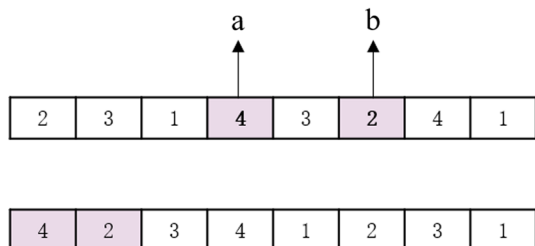
Algorithm 2: OS-VNS

```

Input:  $Iteration_{max}, k_{max}$ 
Output:  $OrderSequence, RackSequence$ 
1 for  $b$  in all PS do
2   Generate a rack sequence  $rseq$  randomly for PS  $b$ ;
3    $iter = 0$ ;
4   while  $iter < Iteration_{max}$  do
5      $k=1$ ;
6     while  $k \leq k_{max}$  do
7       Shaking:
8        $rseq' = Shake(rseq, k)$ ;
9       Local search:
10       $rseq'' = LS(rseq', k_1)$ ;
11       $oseq = OS-Greedy(rseq'')$ ;
12      Move or not:
13      if new solution  $(oseq, rseq'')$  outperforms the best-known solution
14         then
15            $OrderSequence = oseq$ ;
16            $RackSequence = rseq''$ ;
17            $rseq = rseq''$ ;
18            $k = 1$ ;
19         end
20       else
21          $k = k + 1$ 
22       end
23     end
24    $iter = iter + 1$ ;
25 end
    
```

According to the algorithmic procedure, a random rack sequence $rseq$ for each PS is generated at first. Through the shaking procedure, a new rack sequence $rseq'$ is found by the current neighborhood operator k . Then, in the local search stage, an improved rack sequence $rseq''$ for PS b is firstly deduced by applying the local search function, then an order sequence $oseq$ can be obtained by using OS-Greedy on the

Fig. 6 An example of the Hybrid-Relocate operator



improved rack sequence $rseq''$. The next step is to evaluate whether the new solution updates the best-known solution: if yes, update the best-known solution as this new solution, let $rseq''$ be the incumbent rack sequence $rseq$, and resume the neighborhood structure to 1; otherwise, switch to the next neighborhood operator. Solutions are evaluated by counting the rack moves to complete all orders. The shaking, local search and “move or not” steps are repeated until the number of iterations reaches $Iteration_{max}$ and then the algorithm terminates. Note that the solution is immune to inter-PS rack conflict since we have called OS-Greedy (Line 11). The designated neighborhood structures are illustrated as follows.

Neighborhood operators: The purpose of neighborhood operators of the OS-VNS is to slightly change the current rack sequence. Three neighborhood structures 2-Exchange, Segment-Relocate and Hybrid-Relocate are designed, which are illustrated with examples as follows. For each PS, it should be noted that the new rack sequence obtained by a neighborhood operator may contain duplicate racks at adjacent positions which should be avoided. Therefore, if this happens, we revise the sequence by just removing the duplications. In addition, any new generated rack sequence has to be submitted to a conflict elimination procedure. Also, for ease of description, when we randomly select two racks from a rack sequence, they are expressed by a and b respectively.

- 2-Exchange As is illustrated in Fig.4, the positions of two selected racks (4 and 2) are exchanged, and the rack sequence between them (1 5) are reversed to (5 1). At the same time, it can be noticed that the length of the new rack sequence is reduced by 1 as a duplicate rack (4) is removed from it.
- Segment-Relocate As is depicted in Fig.5, racks 3 and 2 are randomly selected, then, the sub-sequence behind rack 2, which contains racks 4 and 1, is inserted into the position between racks 3 and 2.
- Hybrid Relocate Considering the example in Fig.6, the two selected racks (4 and 2) are placed at the head of the new rack sequence, while racks 3, 4, and 1 which are originally behind rack 4 are inserted behind rack 2.

Local Search The local search adopts the first operator.

4.2.3 The rack sequencing algorithms

Similar to OS-Greedy, a greedy algorithm RS-Greedy is also designed to obtain a rack sequence aiming to minimize the number of rack moves under a given order sequence. Again, a metaheuristic algorithm RS-VNS is designed to generate a rack sequence and an order sequence simultaneously.

1. RS-Greedy

RS-Greedy provides a quick approach to generate an appropriate rack visiting sequence at each PS according to a given order sequence. The basic idea is consistent with OS-Greedy: given maximum C parallel orders in each PS, a rack is selected

Table 4 Parameters for instance generation

Parameter	Description	Small instance	Large instance	Sensitivity analysis
N	Number of orders	10,20	50,100	100
M	Number of racks	5,10	50,100	100
K	Number of SKU types	5	10	6-12
S	Number of PS	2,3	2,3,4	1-8
C	Capacity of each PS	3	3	2-18
Q	Queue length	1,2	3	1-5
θ	Maximum number of different SKUs in a rack	4	5	3-5

to visit which has the most common SKUs with all these orders. Here, the rack conflict can be avoided on-the-fly: determine whether there are conflicts between the selected rack and the previously determined rack sequences in other PS; if yes, just choose the next best rack. In this way, we can directly obtain the conflict-free rack sequences. To make it concise, we present the details of RS-Greedy in the Appendix A.

2. RS-VNS

RS-VNS algorithm is to find a rack sequence and an order sequence from the perspective of shuffling the order sequence, aiming to minimize of total rack moves to complete all the orders. RS-VNS is quite similar to OS-RNS, therefore we give very brief introduction here. The pseudocode of RS-VNS is shown in Algorithm 4 in Appendix B. At the beginning of the algorithm, an order sequence $oseq$ is generated randomly for each PS. Then we shuffle the order sequence by shaking procedure to have a new order sequence $oseq'$. After that, in the local search stage, an improved order sequence $oseq''$ is firstly generated by the a local search with operator k_1 , and then a rack sequence $rseq$ is obtained by applying the RS-Greedy on $oseq''$. The next step is to evaluate whether the new solution is updated: if yes, update the best-known solution as this new solution, move incumbent order sequence to $oseq''$, and resume the neighborhood operator to 1; otherwise, switch to the next neighborhood operator. The shaking, local search and “move or not” steps are repeated until the number of iterations reaches $Iteration_{max}$ and then the algorithm terminates. Note also that there is no worry about rack conflict in RS-VNS since the embedded RS-Greedy has already handled this issue. The neighborhood operators are inherited from OS-VNS, that is, 2-Exchange, Segment-Relocate, and Hybrid-Relocate, with exceptions that we change the concept rack to order.

Table 5 Results of CPLEX and four heuristics on small scale instances

<i>S</i>	<i>Q</i>	<i>N</i>	<i>M</i>	CPLEX		OS-Greedy		OS-VNS		RS-Greedy		RS-VNS	
				Obj.	Time	Obj.	Time	Obj.	Time	Obj.	Time	Obj.	Time
2	1	20	10	2	337.79	5.7	0.14	2.9	5.98	6	0.14	2	4.75
		10	5	2	0.48	3.3	0.10	2	2.37	2	0.10	2	1.98
2	2	20	10	3	551.27	5.2	0.14	4	5.86	4	0.14	3	4.67
		10	5	3	0.46	3.9	0.13	3	2.44	3	0.10	3	1.91
3	1	20	10	3	1661.70	4.6	0.14	3	5.57	3	0.14	3	5.49
		10	5	3	1.76	5.2	0.14	3	2.85	3	0.11	3	2.17
2	2	20	10	3	208.45	6.2	0.14	3.2	7.32	3	0.14	3	5.20
		10	5	4	1.20	4.7	0.12	4	2.89	5	0.10	4	2.47
Average				3.1	712.08	4.85	0.13	3.14	4.41	3.62	0.12	3.1	3.58

Table 6 Results of the five heuristics on large instances

N	M	OS-Greedy		OS-VNS		RS-Greedy		RS-VNS			AH		
		Obj.	Time	Obj.	%Impr.	Time	Obj.	Time	Obj.	%Impr.	Time	Obj.	Time
100	100	91.1	0.29	75.6	20.5	64.62	40.0	0.17	28.9	38.4	23.99	29.9	10.19
50	100	53.3	0.24	40.6	31.3	62.22	21.0	0.13	15.6	34.6	16.43	18.3	1.66
50	50	56.3	0.72	47.4	18.8	45.16	25.0	0.12	15.8	58.2	14.17	19.6	6.77
Average		66.9	0.42	54.5	23.5	57.33	28.7	0.14	20.1	43.7	18.20	22.6	6.21

5 Numerical experiments

In this section, we design experiments to evaluate the performance of the proposed algorithms and study the influence of different strategies. We implement the algorithms in Java 8 and run the experiments on a 64-bit win10 system using an Intel Core i5 2.4G CPU and 4G of RAM. To provide a benchmark for the results of the algorithms, CPLEX 12.6 is also used to solve the model built in Sect. 3.

5.1 Instances generation

We design two sets of instances, a small instance set for model and algorithm verification and a large instance set for realistic considerations. Though a big distribution center may need to handle hundreds of thousands of orders per day, we can often handle a small group of orders on a planning horizon. For example, batching orders (according to discretized time) is one such means, and splitting a large order group into small ones according to storage zones can be another means. Therefore, as in Boysen et al. (2017a), we set both orders and racks size as 50 or 100 in our large scale instances. A shared storage policy is applied so that K SKUs randomly scatter on M racks. The number of SKU types in each rack is generated uniformly in $[1, \theta]$, where $\theta < K$ reflects the degree of dispersion of SKUs on racks. The number of SKU types contained on each order follows a Poisson distribution with a mean of λ_i , $i \in \{1, 2, \dots, K\}$, where λ_i corresponding to each SKU i is independent of each other. To accommodate diversity of orders, λ_i is set to obey the uniform distribution of $[0.2K, 0.8K]$. Table 4 presents the parameters in the experiments. We generate ten instances for each combination of parameters and take the average results.

5.2 Algorithmic performance

In the proposed two-stage hybrid heuristic algorithm, we have adopted two order assignment strategies in the first stage: Random strategy and Order Similarity strategy. Also, in the second stage, we propose two metaheuristics OS-VNS and RS-VNS from two perspectives, both of which are capable of generating an order sequence and a rack sequence simultaneously. Furthermore, the proposed two greedy algorithms OS-Greedy and RS-Greedy can also act as a solution approach to the Secondhandshop-stage problem if they are fed with a randomly-generated rack sequence or order sequence. Therefore, we implement four heuristics OS-Greedy, OS-VNS, RS-Greedy and RS-VNS on small and large instances. Meanwhile, we solve small-scale instances with CPLEX to optimal, the results of which are used as a reference to evaluate the proposed algorithms. Throughout the experiments, all heuristic and metaheuristic algorithms are run 10 times and the averages are reported as the final results. For OS-VNS and RS-VNS, parameter $Iteration_{max}$ takes value 100.

The results for small scale instances are listed in Table 5 in which the objective value (Obj. in the table) and CPU time (in second) are reported for 5 solution approaches: CPLEX, OS-Greedy, OS-VNS, RS-Greedy and RS-VNS. The

Table 7 Comparison of different strategies by using RS-VNS

S	N	M	RANDOM		SIMILARITY		%Impr.3	%Impr.4		
			SINGLE	MULTIPLE	%Impr.1	%Impr.2				
									SINGLE	MULTIPLE
2	100	100	36.3	31.8	14.15	36.2	31	16.77	0.28	2.58
	50	100	19.6	14.3	37.06	19	14.1	34.75	3.16	1.42
	50	50	21.9	15.6	40.38	21	15.1	39.07	4.29	3.31
3	100	100	39.1	31.2	25.32	37.3	28.9	29.07	4.83	7.96
	50	100	20.2	14.8	36.49	20.1	15.7	28.03	0.50	-5.73
	50	50	22.7	15.5	46.45	22.9	16.4	39.63	-0.87	-5.49
4	100	100	39.9	31.7	25.87	37.6	30.5	23.28	6.12	3.93
	50	100	20.8	16.6	25.30	21.1	16.6	27.11	-1.42	0.00
	50	50	24.6	17.3	42.20	24.4	17.2	41.86	0.82	0.58
Average			27.2	21.0	32.58	26.6	20.6	31.06	2.30	1.78

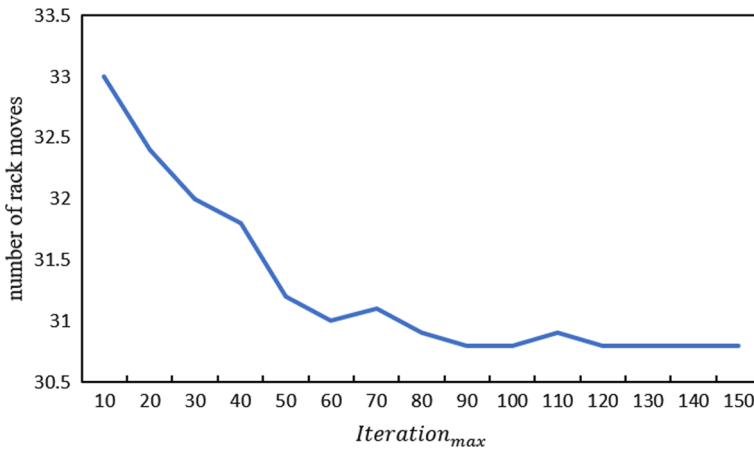


Fig. 7 Convergence of the proposed VNS-based algorithms

order assignment strategy used here is Order Similarity and queue length Q of 1 and 2 are tested. When observing the objective value, it shows that RS-VNS can obtain optimal solutions in all cases which proves its effectiveness for small scale instances. RS-Greedy is also capable of obtaining optimal solution on 5 of the 8 instances. It seems OS-Greedy algorithm is too coarse and fails to provide promising solutions. OS-VNS performs slightly worse than RS-VNS and it takes more CPU time, however, it improves its initial solution, that is, OS-Greedy, to a large extent. Furthermore, when we compare the computational time of the 5 solution approaches, we can easily find out the CPLEX is very time-consuming as the scale increases, while the proposed heuristics and metaheuristics show high computing efficiency.

Since CPLEX cannot solve larger problems, we use OS-Greedy, OS-VNS, RS-Greedy, and RS-VNS to solve large-scale instances. To facilitate a comparison with the previous study, we also implement an Alternating Heuristic (AH) proposed in Boysen et al. (2017b). AH is invoked after the order assignment stage. It begins with an initial rack, chooses C orders from that PS according to their similarity, and picks all items these orders need from that rack. Then, for those remaining items, choose an appropriate rack based on the similarity again, and replenish orders if the number of in-process orders is less than C . The process repeats until all orders are fulfilled. The results of the five heuristics are shown in Table 6.

It shows that RS-Greedy and RS-VNS outperform the OS-Greedy and OS-VNS in all instances which is consistent with the results in small instances. It can be concluded that the algorithms based on rack sequence perform better than that based on order sequence in terms of both the number of rack moves and CPU time. Both RS-VNS and OS-VNS improve the results of their greedy counterparts significantly by 23.52% and 43.75%, respectively, proving the effectiveness of the proposed VNS-based procedures. It is interesting to see that the AH algorithm behaves much better than others but still worse than RS-VNS. The average objective value by AH is 22.6, 12% higher than RS-VNS. AH takes much less time

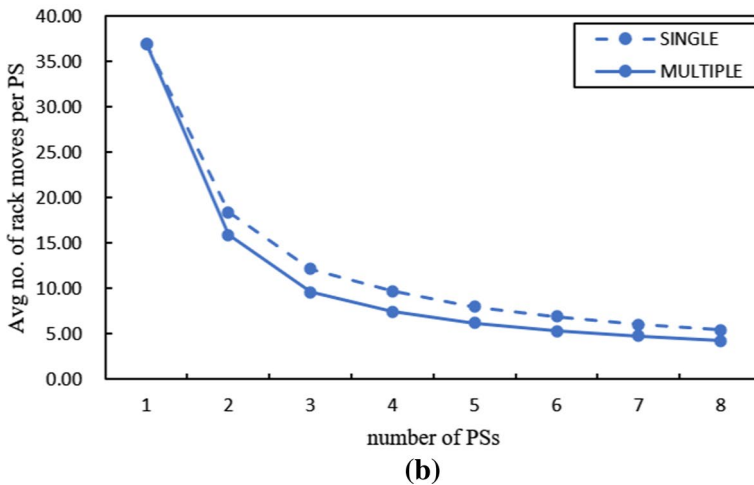
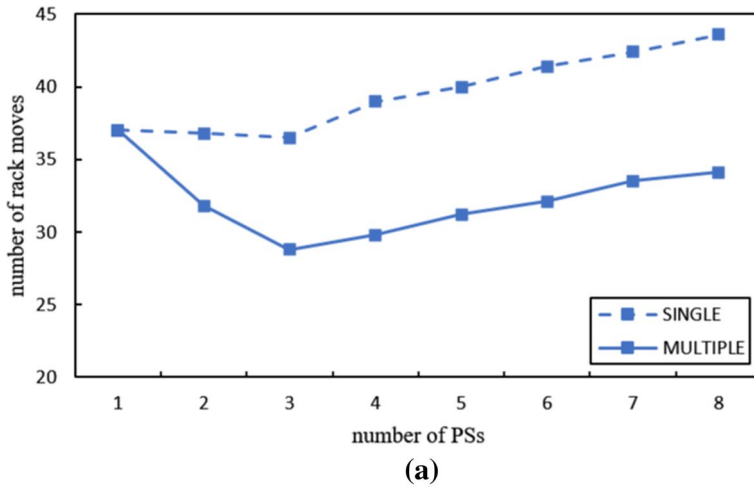


Fig. 8 Effect of the number of PS on rack moves

than RS-VNS since it does not need a neighborhood search. When it is a large scale problem, AH is also a good choice.

From Table 6 we also observe that, for the same number of orders, the more racks available for picking, the fewer the number of rack moves required to complete the orders. As shown in Table 6, when the number of orders is 50, to complete all the orders, the number of rack moves with 100 racks is less than that with 50 racks.

As RS-VNS shows better performance, it is used for the next experiments in which two strategies are investigated: the order assignment strategy and the rack visiting strategy. In this paper, we assign orders to PS according to either Random

strategy (denoted as RANDOM) or Order Similarity strategy (denoted as SIMILARITY). Meanwhile, we allow a rack to serve another PS immediately after serving the current one when it is necessary, as opposed to the traditional way in which a rack visits just one PS each time. The two rack visiting strategies are denoted as MULTIPLE and SINGLE, respectively. This setting of experiments is based on large scale instances in which the number of PS varies from 2 to 4. Table 7 reports the results (objective values) where $\%Impr.1$ and $\%Impr.2$ are the improvements in percentage of the objective value (rack moves) deduced by the MULTIPLE policy to that by the SINGLE policy, while $\%Impr.3$ and $\%Impr.4$ represent the improvement by the SIMILARITY method to that by the RANDOM method, according to SINGLE and MULTIPLE policies respectively. Note that positive improvement percentage means reduction of rack moves.

By reviewing Table 7, we can find out the following facts. First, less rack moves are observed by strategy MULTIPLE than by SINGLE. For example, the average $\%Impr.1$ (for RANDOM policy) is 32.58% and the average $\%Impr.2$ (for SIMILARITY policy) is 31.06%, exhibiting a large reduction of rack moves. This validates the hypothesis that allowing movable racks to visit multiple PS in one trip can lead to noticeable savings in order processing. Second, it can be seen that assigning orders to PS according to the SIMILARITY strategy can reduce the number of rack moves compared to randomly assigning orders to PS. The improvement $\%Impr.3$ by using SINGLE strategy is 2.30%, and $\%Impr.4$ by MULTIPLE strategy is 1.78%. Although the advantage of similarity-based assignment is not significant, it is also not neglectable.

Finally in this section, we also dig into the calculation process to examine the convergence of the proposed VNS-based algorithms. Take $N = 100$, $M = 100$, $S = 3$ and RS-VNS as an example, the evolution of the number of rack moves over iteration

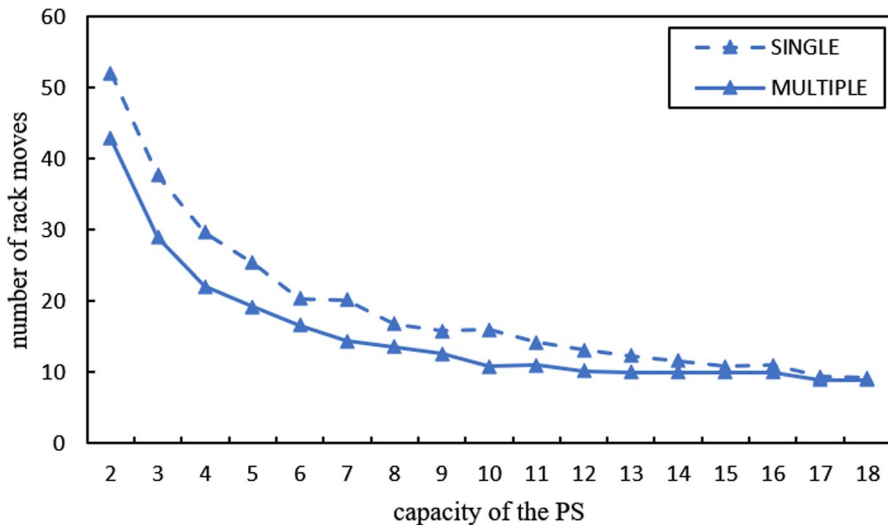


Fig. 9 Effect of the capacity of PS on rack moves

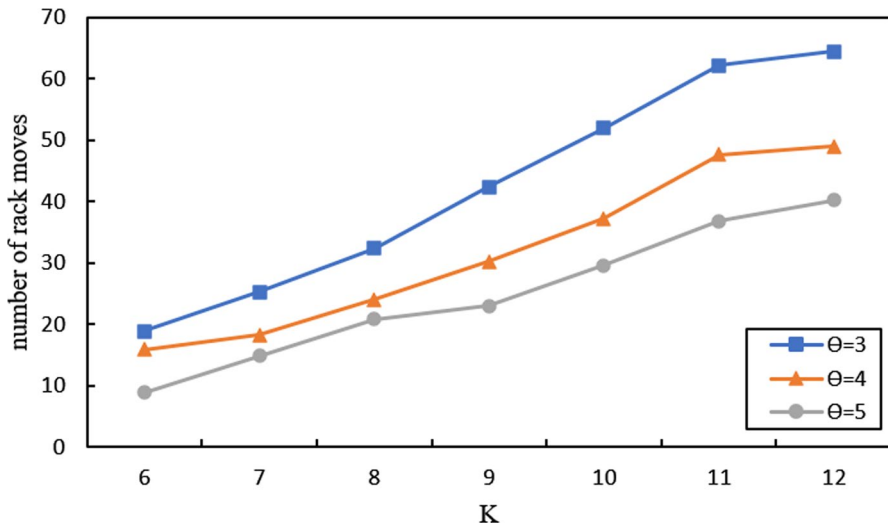


Fig. 10 Effect of the types of SKUs on rack moves

times is shown in Fig. 7. It seems that the objective tends to be steady after 120 iterations, therefore, we set $Iteration_{max} = 120$. Likewise, we take $Iteration_{LS} = 10$ by experiments to achieve a balance between accuracy and computational time.

5.3 Sensitivity analysis

In this section, sensitivity analysis are conducted to study the effect of different parameters used in the proposed algorithms. We design the next experiments in which the number of orders is 100 and the number of racks is 100. In the experiments, the customer orders are assigned to PS based on the Order Similarity strategy. The parameters that we focus on contain: (i) the number of PS (S); (ii) the capacity of each PS (C); (iii) the types of SKUs (K); (iv) the queue length (Q). Throughout all sensitivity analysis, RS-VNS is used due to its superior performance.

5.3.1 Effect of the number of PS

We investigate the impact of the number of PS by varying it from 1 to 8 while fixing other parameters as $N = 100$, $M = 100$, $K = 10$, $C = 3$, $Q = 3$ and $\theta = 6$. Meanwhile, both SINGLE and MULTIPLE rack visiting strategies are considered. The number of rack moves for different number of PS are presented in Fig. 8a in which the dashed line represents the SINGLE strategy while the solid line represents the MULTIPLE strategy. According to the strategy of MULTIPLE and the procedure of the sequence improvement in the proposed algorithm, the gap between MULTIPLE and SINGLE policies actually stands for the rack moves between PS. It shows that the gap increases as S increases which is reasonable since there would be more rack moves between PS replacing the rack transports between the storage area and the

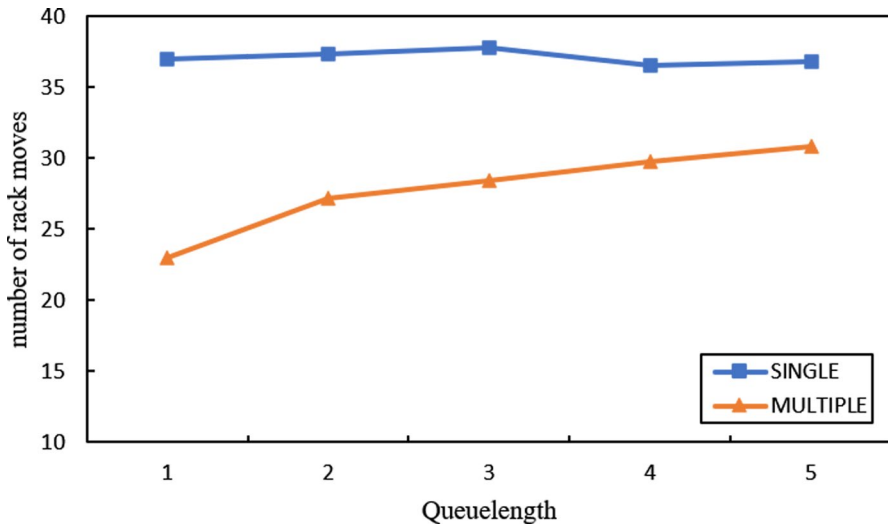


Fig. 11 Effect of the queue length on rack moves

picking area when more PS are available. It can also be found that the number of rack moves by both strategies tends to decrease first and then increase slightly as S increases. It shows that involving too many PS, for example, 7 or 8, will on the contrary increase the rack moves.

To better reflect the throughput time of the given orders, we present the average rack moves per PS in Fig. 8b in which a clear trend can be observed that the average number of rack moves decreases as S increases. However, the slope of the curves in Fig. 8b is very steep at first and turns very flat when the number of PS goes beyond 4. Since more PS incur more human labors, more space requirement and other potential costs, hence, the number of PS needs to be set up carefully. In the following sensitivity analysis experiments, the number of PS defaults to 3.

5.3.2 Effect of the capacity of each PS

The capacity of each PS C is supposed to affect the necessary rack moves of completing all orders: larger C means more SKUs could be retrieved by a single rack visit and therefore the total rack moves can be reduced. We take sensitivity analysis by varying C from 2 to 18 while fixing other parameters, for example, $N = 100$ and $M = 100$. Both SIMPLE and MULTIPLE strategies are implemented. As can be seen from Fig. 9, both strategies exhibit the same varying trends with the PS capacity. The total number of rack moves required to complete all orders decreases as C increases, as expected. However, the trend shows that the decreasing rate is getting smaller and smaller in general, indicating that the marginal benefits brought by increased C is gradually fading. When C increases to a certain level, such as 10, the number of rack moves tends to stabilize. Actually, limited by the space of PS,

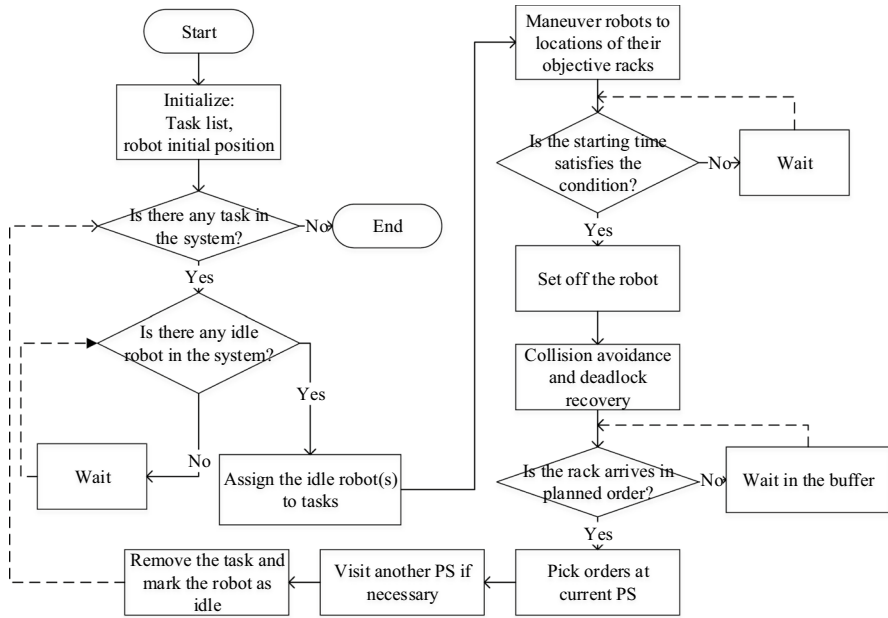


Fig. 12 The simulation framework

C cannot be increased infinitely. Meanwhile, larger C also adds inconvenience to the human pickers as they need to take care of more units. Therefore, in reality, the capacity should be set as larger while considering the human factors.

5.3.3 Effect of the SKU diversity

We investigate the effect of the SKU diversity on the RMFS efficiency. Specifically, we consider both the total number of SKU types in the RMFS (K), and the number of SKU types in each rack which is uniformly distributed in $[1, \theta]$. Based on the large scale instance $N = 100$, $M = 100$ and $S = 3$, experiments are conducted by setting $K \in \{6, 7, \dots, 12\}$ and $\theta \in \{3, 4, 5\}$ and the results are presented in Fig. 10. As K increases, the total number of rack moves increases in an approximately linear manner, regardless of the value of θ . It indicates that the number of SKU types has a positive correlation with the order picking workload. On the other hand, when observing the performance of $\theta=3, 4$ and 5 presented in Fig. 10, it shows that the more types of SKUs each rack contains, the fewer rack moves required to complete the orders. Hence, increasing the SKU diversity of the racks can effectively reduce the number of rack moves. It should be mentioned also that the shared storage policy will incur more efforts in the replenishment process, therefore, an integrated study

covering complete order processing cycle is meaningful in the future, see for example, Roy et al. (2019).

5.3.4 Effect of the queue length

Finally, we investigate the effect of the queue length Q , which restricts the positions of the same rack in different rack sequences. As we know, queue of racks before a PS helps save the picker's waiting time for racks, however, will the queue help reduce the total rack moves? Our answer is no when inter-PS service is permitted, after we conduct the following experiments. We record the number of rack moves by varying Q from 1 to 5 based on large instance $N = 100$, $M = 100$ and $S = 3$, as shown in Fig. 11. It reveals a fact that the total number of rack moves required to complete all orders is not sensitive to parameter Q under the SINGLE strategy. However, it tends to increase as Q increases under the MULTIPLE strategy. Another interesting fact can be observed from Fig. 11 that the gap between the two curves is getting smaller as Q increases, which means that the number of rack moves between PS decreases as Q rises. The phenomenon can be intuitively explained as: when Q rises, there are more racks queueing in the front of PS which are prohibited to serve other PS, therefore, less inter-PS rack moves can occur, as a result, the rack moves between the picking area and the storage area increases. Therefore, as a managerial insight, we are suggested to use shorter queue length in the RMFS as long as the queue of racks can supply the human picker in time.

6 Simulation study

Until now the objective of the proposed optimization problem and the proposed two-stage hybrid algorithm is to minimize the total number of rack moves between the storage area and the PS, however, it seems more realistic and intuitive to consider the completion time of all orders as the measure of the RMFS performance. We also took a bold assumption that the length of a time slot in different PS is the same, however, the violation of this assumption may affect the throughput time of the order picking process. In this study, we design an event-driven simulation system from scratch using Java (see also Duan et al. (2021)), and find that the two goals behave consistently, proving that the assumption on the time slot length is acceptable. The framework of the simulation is shown in Fig. 12, in which the following steps are followed:

Table 8 Regression coefficients

Number of robots	a_0	a_1	a_2	a'_1	a'_2	confidence level
3	34.3	30.5	22.5	0.9867	0.0412	0.99
6	32.7	15.8	10.3	0.9887	0.0369	0.99
9	41.52	10.73	2.75	0.9920	0.0143	0.99

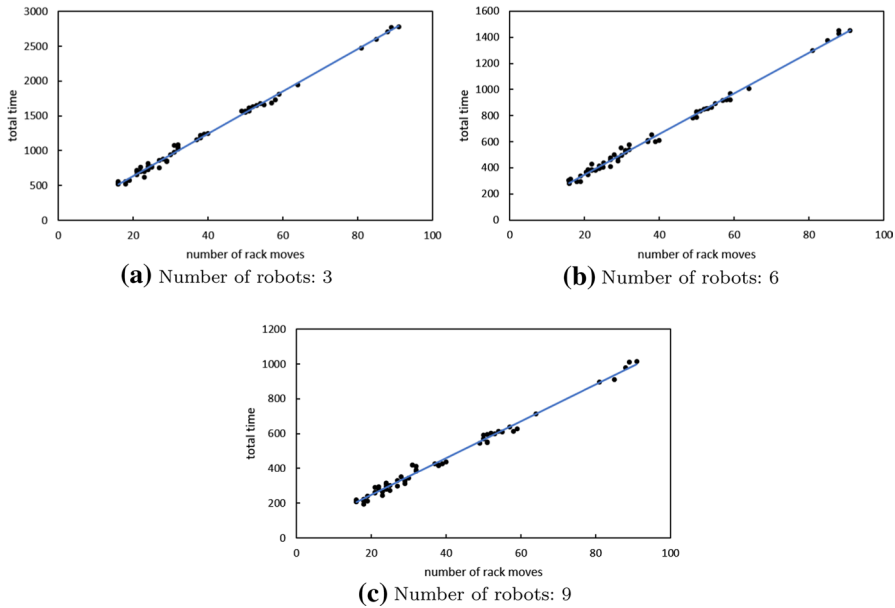


Fig. 13 Correlation between the number of rack moves and the completion time

(1) Tasks and robots initialization. A task corresponds to an element in a rack sequence which is defined by a rack number and its corresponding PS number indicating which rack should be carried to which PS. At the beginning of the experiment, a list of tasks is initialized according to the rack sequences obtained by the proposed RS-VNS algorithm. The first tasks of all PS is firstly added to the task list, then followed by the second ones, and so on so forth. A fleet of robots is initially waiting at the parking area on an idle status.

(2) Task assignment. Whenever there are idle robots, tasks are extracted from the task list in order and assigned to a nearest idle robot. When a robot is associated with a task, it then moves to the origin of the task, that is, the storage location of the rack. The A* algorithm is used for the path planning of the robots to generate shortest routes. In a multiple-robot environment, collision between robots must be avoided and deadlocks should be avoided or resolved once happen. In this study, we adopt an authority-based mechanism to avoid collisions, and a deadlock detection and recovery method to handle the deadlocks.

(3) Set off the robots. When a robot is assigned a task and arrives at the objective rack, it may not set off immediately since it must not arrive at the PS earlier than the task before it. For a task i , denote its departure time from the objective rack as T_i^{dep} , and denote the shortest travel time from the rack to the PS as T_i^{travel} , then the departure time of task i should be $T_i^{dep} = \max\{current\ time, T_{i-1}^{dep} + T_{i-1}^{travel} - T_i^{travel}\}$, where task i and task $i - 1$ belong to the same PS.

(4) Service at the PS. Considering that there may need extra time for collision avoidance and deadlock recovery in the process of robots moving toward the PS,

some racks may not arrive at the corresponding PS in the planned order if no action is taken. We set buffers for each PS, and the rack which arrives earlier than originally planned can wait in the buffer until its previous rack arrives. In this way, the arrival sequence can be strictly guaranteed. When a rack is visiting the PS, orders extracted from the order sequence are fulfilled and finished orders are removed from the system.

(5) Inter-PS visit. Before completion of the current task, the robot will be assigned with another task if it is going to perform an inter-PS service in the next step.

(6) Return to the storage area. After servicing the PS, a rack will be returned to a nearest empty storage area.

The warehouse layout in the simulation is similar to Fig. 1 which is about 1200 m^2 in size containing three PS ($Q = 3$). In the storage area, there are 360 rack locations in total, separated by 5 aisles and 1 cross aisle. Each rack location is $1.2 \times 1.2\text{ m}^2$. The average speed of the robots is 1 m/s and we ignore the acceleration and deceleration. In the simulation experiments, we randomly-generated 5 instances of scale ($N = 50, M = 50$), ($N = 50, M = 100$) and ($N = 100, M = 100$), and solve them with algorithms OS-Greedy, OS-VNS, RS-Greedy, and RS-VNS, respectively. Therefore, we have 60 solutions (rack sequence and order sequence for each PS) which are fed into the simulation system as input. To accommodate different robot fleet size, we apply 3, 6 and 9 robots to the simulation system, respectively, to deduce the total completion time which is measured by the time when the last order is fulfilled. Therefore, for each robot number, we can regress a function in which the number of rack moves (x_1) and number of inter-PS services (x_2) are independent variables, and the total completion time (y) is the dependent variable. We take a linear regression function as $y_i = a_{i0} + a_{i1}x_{i1} + a_{i2}x_{i2}$ where $i \in \{1, 2, 3\}$ represents conditions when 3, 6, and 9 robots are used. After obtaining the regression coefficients $a_{ij}, i \in \{1, 2, 3\}, j \in \{0, 1, 2\}$, we further calculate the standardized regression coefficients $a'_{ij}, i \in \{1, 2, 3\}, j \in \{1, 2\}$ since each of them refers to how many standard deviations the completion time will change, per standard deviation increase in the predictor variable. The results are shown in Table 8.

From Table 8 we can find out that the regression is in good quality as the confidence level of the three groups of experiments is as high as 99%. It can also be noticed that the standardized regression coefficient of x_1 is much larger than that of x_2 . Therefore, the number of inter-PS rack moves has a very limited effect on the completion time, therefore can be ignored. To better reflect the relationship between the number of rack moves and the completion time, we generate the scatter plots as shown in Fig. 13, which clearly suggests a linear function between them. Therefore, the simulation proves that to consider the number of rack moves as an objective is reasonable.

7 Conclusions

This paper studies the order picking problem in a robotic mobile fulfillment system with multiple picking stations, which jointly addresses the order assignment, order sequencing, rack selection, and rack sequencing subproblems. The inter-PS rack service and the queueing effect of each PS are specifically considered. We formulate this problem as an integer programming model to minimize the number of rack moves. Then, a two-stage hybrid heuristic algorithm framework is proposed to solve the problem. Specifically, we design algorithms OS-Greedy and OS-VNS from the perspective of order sequencing and algorithms RS-Greedy and RS-VNS from rack sequencing. Numeric experiments show that RS-VNS outperforms other algorithms. A significant finding is that allowing inter-PS service can improve the solution by 31.06% on the testing cases. This finding encourages the industry to adapt to this picking mode rather than operating independent PS orders.

We also conduct a series of sensitivity analysis to draw managerial insights. For example, more PS tend to decrease the completion time, however, marginal benefit decreases as the number of PS increases. Likewise, larger PS capacity can also help decrease the number of rack moves, however, human factors must be considered at the same time. The total number of SKU types K has been observed to have an approximately linear and positive effect on the number of rack moves, while the scattering of SKUs to more racks can effectively decrease the rack moves. An interesting fact about queue length Q is that the objective is not sensitive to Q under the SINGLE strategy, while it tends to increase as Q increases under the MULTIPLE strategy. Therefore, we are suggested to use shorter queue length in the RMFS as long as the queue of racks can supply the human picker in time. Finally, the simulation system verifies that the total completion time is linearly related to the number of rack moves.

In this study, also in Boysen et al. (2017a) and Valle and Beasley (2021), the objective function is represented by the number of times that racks travel between PS and the storage area. In future work, one can consider how to formulate the problem to account for the actual transportation distance and time. Besides, research efforts can also be devoted to considering different storage policies, for example, the class-based storage policy and the dedicated storage policy.

Appendix A RS-Greedy

We denote the difference between rack j and the current in-process orders as $\text{DiffVector}(j)$, then the pseudocode of the RS-Greedy is depicted in Algorithm 3 where PickingOrderSet , RemainingOrderSet and DiffMatrix have the same meaning as in OS-Greedy, and RackSet is the set of all racks. According to the algorithmic procedure, orders are extracted from the given OrderSequence successively, while the rack with the minimum difference with all the in-process orders (excluding those finished SKUs) and free of conflict is selected to visit the PS one by one. Completed orders will be removed and replaced by new orders. Until the SKUs contained on

the current picking orders cannot be fulfilled anymore by the current rack, a new rack will be selected, so on so forth.

Algorithm 3: RS-Greedy

Input: *OrderSequence*
Output: *RackSequence*

```

1 for  $b$  in all PS do
2   Initiate PickingOrderSet as the first  $C$  orders of OrderSequence( $b$ );
3   Initiate RemainingOrderSet as the last  $N - C$  orders of OrderSequence( $b$ );
4   while the number of completed orders  $< N_b$  do
5     for  $j$  in RackSet do
6        $DiffVector(j) = 0$ ;
7       for  $i$  in PickingOrderSet do
8          $DiffVector(j) = DiffVector(j) + DiffMatrix(i, j)$ ;
9       end
10    end
11    Select rack  $r$  such that  $r = \operatorname{argmin}_{j=1,2,\dots,M} DiffVector(j)$  and  $r$  is
        conflict-free;
12    Append  $r$  to RackSequence( $b$ );
13    Pick items from rack  $r$ ;
14    Remove finished SKUs from their orders;
15    if any picking order is completed then
16      Remove the order from PickingOrderSet;
17    end
18    while  $|PickingOrderSet| < C$  do
19      Let  $num = \min(C - \text{the number of orders in picking,}$ 
        —RemainingOrderSet—);
20      Remove the first  $num$  orders from RemainingOrderSet and add them
        to PickingOrderSet;
21      Pick items from rack  $r$  for newly-added orders;
22      Remove finished SKUs from their orders;
23      if any picking order is completed then
24        Remove completed orders from PickingOrderSet;
25      end
26    end
27  end
28 end

```

Appendix B Pseudocode of RS-VNS

Algorithm 4: RS-VNS

Input: $Iteration_{max}$, k_{max}

Output: $OrderSequence$, $RackSequence$

```

1 for  $b$  in all PS do
2   Generate an order sequence  $oseq$  randomly;
3    $iter = 0$ ;
4   while  $iter < Iteration_{max}$  do
5      $k = 1$ ;
6     while  $k \leq k_{max}$  do
7       Shaking:
8        $oseq' = \text{Shaking}(oseq, k)$ ;
9       Local search:
10       $oseq'' = \text{LS}(oseq', k_1)$ ;
11       $rseq = \text{RS-Greedy}(oseq'')$ ;
12      Move or not:
13      if new solution ( $rseq, oseq''$ ) outperforms the best-known
14      solution then
15         $OrderSequence = oseq''$ ;
16         $RackSequence = rseq$ ;
17         $oseq = oseq''$ ;
18         $k = 1$ ;
19      end
20      else
21         $k = k + 1$ 
22      end
23    end
24     $iter = iter + 1$ ;
25  end

```

Funding This work is supported by the National Key R&D Program of China under grant No. 2018AAA0101705, the National Natural Science Foundation of China under Grant No. 71772100, and Shenzhen Science and Technology Project under Grant No. JCYJ20170412171044606, and Sichuan Science and Technology Program under grant No. 2021JDRC0009.

References

Azadeh K, De Koster R, Roy D (2019) Robotized and automated warehouse systems: review and recent developments. *Transp Sci* 53(4):917–945

- Bartholdi III JJ, Hackman ST (2008) Warehouse and distribution science. Supply Chain and Logistics Institute, School of Industrial and Systems Engineering, Georgia Institute of Technology. https://www.scl.gatech.edu/sites/default/files/downloads/gtsclwarehouse_science_bartholdi.pdf
- Boysen N, Briskorn D, Emde S (2017) Parts-to-picker based order processing in a rack-moving mobile robots environment. *Eur J Opera Res* 262(2):550–562
- Boysen N, Briskorn D, Emde S (2017) Sequencing of picking orders in mobile rack warehouses. *Eur J Opera Res* 259(1):293–307
- Boysen N, Fedtke S, Weidinger F (2018) Optimizing automated sorting in warehouses: the minimum order spread sequencing problem. *Eur J Opera Res* 270(1):386–400
- Boysen N, de Koster R, Weidinger F (2019) Warehousing in the e-commerce era: a survey. *Eur J Opera Res* 277(2):396–411
- Bozer YA, Aldarondo FJ (2018) A simulation-based comparison of two goods-to-person order picking systems in an online retail setting. *Int J Prod Res* 56(11):3838–3858
- De Koster R, Le-Duc T, Roodbergen KJ (2007) Design and control of warehouse order picking: a literature review. *Eur J Opera Res* 182(2):481–501
- Duan G, Zhang C, Gonzalez P, Qi M (2021) Performance evaluation for robotic mobile fulfillment systems with time-varying arrivals. *Comp Ind Eng* 158:107365
- Füßler D, Boysen N (2019) High-performance order processing in picking workstations. *EURO J Transp Logist* 8(1):65–90
- Grosse EH, Glock CH, Jaber MY, Neumann WP (2015) Incorporating human factors in order picking planning models: framework and research opportunities. *Int J Prod Res* 53(3):695–717
- Gu J, Goetschalckx M, McGinnis LF (2007) Research on warehouse operation: a comprehensive review. *Eur J Opera Res* 177(1):1–21
- Huang GQ, Chen MZ, Pan J (2015) Robotics in ecommerce logistics. *HKIE. Transactions* 22(2):68–77
- Jagheer Y, Hanson R, Johansson MI (2020) Automated order picking systems and the links between design and performance: a systematic literature review. *Int J Prod Res* 58(15):4489–4505
- Lamballais T, Roy D, De Koster MBM (2019) Inventory allocation in robotic mobile fulfillment systems. *IIESE Transactions* pp 1–17
- Li Z, Li W (2015) Mathematical model and algorithm for the task allocation problem of robots in the smart warehouse. *Am J Opera Res* 5(06):493
- Li ZP, Zhang JL, Zhang HJ, Hua GW (2017) Optimal selection of movable shelves under cargo-to-person picking mode. *Int J Simul Modell* 16(1):145–156
- Menéndez B, Pardo EG, Alonso-Ayuso A, Molina E, Duarte A (2016) Variable neighborhood search strategies for the order batching problem. *Comp Opera Res* 78:500–512
- Merschformann M, Lamballais T, Koster MBMD, Suhl L (2019) Decision rules for robotic mobile fulfillment systems. *Opera Res Perspect* 6:100128
- Mladenović N, Hansen P (1997) Variable neighborhood search. *Comp Opera Res* 24(11):1097–1100
- Öncan T (2015) Milp formulations and an iterated local search algorithm with tabu thresholding for the order batching problem. *Euro J Opera Res* 243(1):142–155
- QuickTron (2020) Quicktron. <http://www.flashhold.com/english.php>
- Roy D, Nigam S, de Koster R, Adan I, Resing J (2019) Robot-storage zone assignment strategies in mobile fulfillment systems. *Transp Res Part E: Logist Transp Rev* 122:119–142
- Scholz A, Schubert D, Wäscher G (2017) Order picking with multiple pickers and due dates-simultaneous solution of order batching, batch assignment and sequencing, and picker routing problems. *Euro J Opera Res* 263(2):461–478
- Tanimoto TT (1957) Ibm internal report. Nov 17:1957
- Valle CA, Beasley JE (2021) Order allocation, rack allocation and rack sequencing for pickers in a mobile rack environment. *Computers & Operations Research* 125:105090
- Weidinger F, Boysen N, Briskorn D (2018) Storage assignment with rack-moving mobile robots in kiva warehouses. *Transp Sci* 52(6):1479–1495
- Wurman PR, D'Andrea R, Mountz M (2008) Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magaz* 29(1):9–9
- Xi X, Liu C, Miao L (2018) Storage assignment and order batching problem in kiva mobile fulfillment system. *Eng Optim* 50(11):1941–1962
- Yuan R, Graves SC, Cezik T (2019) Velocity-based storage assignment in semi-automated storage systems. *Prod Opera Manag* 28(2):354–373
- Zhang J, Yang F, Weng X (2019) A building-block-based genetic algorithm for solving the robots allocation problem in a robotic mobile fulfillment system. *Mathematical Problems in Engineering* 2019
- Zou B, Gong YY, Xu X, Yuan Z (2017) Assignment rules in robotic mobile fulfillment systems for online retailers. *Int J Prod Res* 55(20):6175–6192

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Bingqian Wang is a master student at Tsinghua University, Shenzhen International Graduate School. Her research interests focus on the optimization of logistics systems.

Xiuqing Yang is a researcher at the Civil Aviation Logistics Technology Co., Ltd. Her research interest is technical innovation in aviation logistics.

Mingyao Qi is currently an Associate Professor in Industrial Engineering with Tsinghua University, Shenzhen International Graduate School (SIGS). His research interests include modeling, optimization, and simulation of logistics systems.

Authors and Affiliations

Bingqian WANG¹ · Xiuqing YANG² · Mingyao QI³ 

Bingqian WANG
17888841403@163.com

Xiuqing YANG
yangxiuqing@caltco.com

- ¹ Research Center on Modern Logistics, Shenzhen International Graduate School, Tsinghua University, Shenzhen 518055, China
- ² The Civil Aviation Logistics Technology Co., Ltd., The Second Research Institute of Civil Aviation Administration of China, Chengdu 610041, China
- ³ Research Center on Modern Logistics, Shenzhen International Graduate School, Tsinghua University, Shenzhen 518055, China