Hindawi
Mathematical Problems in Engineering
Volume 2021, Article ID 6626569, 16 pages
https://doi.org/10.1155/2021/6626569

Research Article

A Two-Stage Path Planning Method for the Rearrangement Problem in Modular Puzzle-Based Storage System

Penghui Guo, Fei Xiao, Wanzhi Rui, Zhengrong Jia, and Jin Xu

National Key Laboratory of Vessel Integrated Power System Technology, Naval University of Engineering, Wuhan 430033, China

Correspondence should be addressed to Wanzhi Rui; afeimeng@126.com

Received 8 December 2020; Revised 15 February 2021; Accepted 27 March 2021; Published 23 April 2021

Academic Editor: Dongsheng Guo

The modular puzzle-based storage system consists of multiple storage modules. The system has multiple input/output (IO) points which can simultaneously deal with multiple orders in one batch. When a batch of orders comes in advance, it is necessary to rearrange the items near each IO point to reduce the picking time of customers. To complete the rearrangement quickly, this paper proposes a two-stage path planning method considering the simultaneous movement of multiple items. This method includes two stages: planning single moves and merging single moves. In the first stage, the sequence of single moves of each module needs to be obtained so as to convert the system from an initial state to a target state. In the second stage, the single moves in the sequence are merged into block moves and parallel moves to reduce the steps of movement. The simulation results show that the single move planning method can be used to solve the rearrangement problem stably and effectively and that the single move merging method can greatly optimize the experimental results with the optimization rate more than 50% in different configurations.

1. Introduction

With the rapid development of e-commerce and manufacturing, the automated warehousing system has become a vital link of modern logistics systems and supply chain [1, 2]. Traditional warehousing systems such as automated storage and retrieval systems (AS/RS) generally contain aisles for the vehicle to move and transport items. These systems have been extensively researched by scholars [3–5]. In these systems, the aisles occupy the space that could be used to store items, thereby limiting the storage density. In some fields of application, there are higher demands for storage density and space utilization of storage systems, for example, the ship storage systems and the automated parking systems.

To achieve high-density storage, a so-called puzzle-based storage (PBS) system [6] has been developed in recent years. Different variants of PBS systems have been used in warehouses and distribution centers, automated parking systems, container handling in port terminals, and so on [7–10].

The PBS systems consist of many adjacent square storage cells arranged in a rectangular grid. Each cell either stores an item or is empty, and the item may only be moved from its current cell to its adjacent empty cell (referred to as an escort) (see Figure 1). This process of moving a single item to its adjacent empty cell is called a single move. In some models of PBS systems, multiple items are allowed to be moved simultaneously in a step, called the merged move, which means merging multiple single moves.

As shown in Figure 2, the merged move includes block move and parallel move. The block move means that a series of items arranged adjacently on a straight line, or a block of items can be moved to an adjoining escort. The parallel move means that multiple single moves or block moves can be performed in one step, if their pathways are not in conflict with each other. Obviously, compared with the single move, the merged move can save steps and improve the efficiency of the PBS system. However, the scheme for the merged move that is directly considered in the planning stage will make the computation more complicated, and it is infeasible.

In recent years, multiple variants of the PBS system have been studied. Gue and Kim [6] were the first to propose the concept of the PBS system that was an inspiration from the

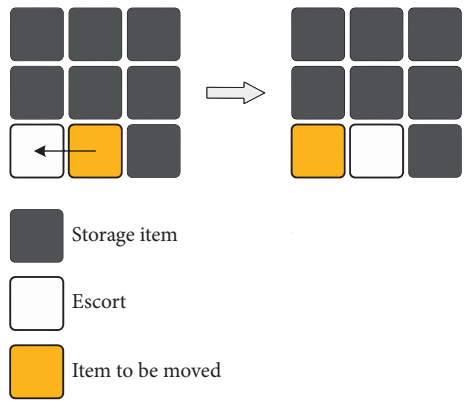


Figure 1: The moving process of a PBS system.

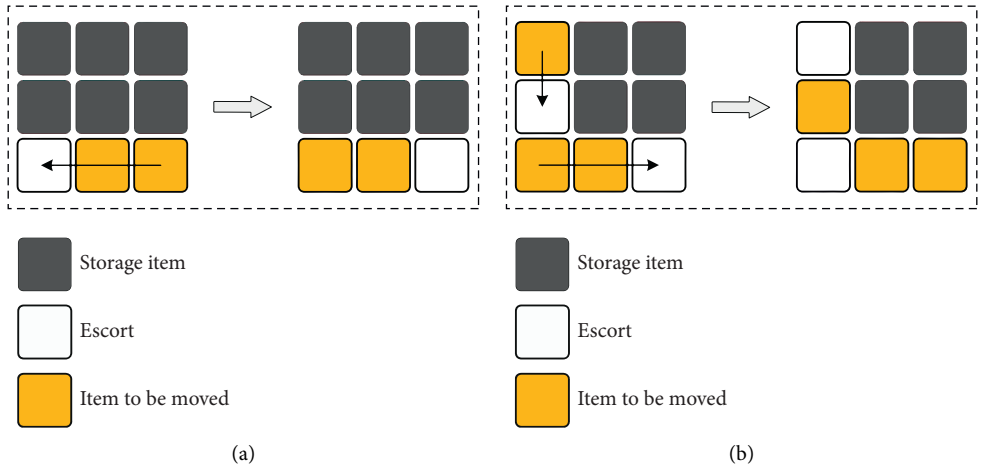


Figure 2: Block move and parallel move. (a) Case of block move. (b) Case of parallel move

famous 15-puzzle game. They provided optimal solutions to retrieve one item in a PBS system with one escort in the lower left corner which is the location of the IO point. For the larger problems with multiple escorts arranged in a straight line near the IO point, they proposed a heuristic method. In addition, the block move was taken account in the model established by them. The limitation of their work is that one or more escorts must be located near the IO point in the initial state. Kota et al. [11] proposed a general solution for a PBS system with multiple escorts using integer programming (IP). This approach seems to be impractical due to its poor computational efficiency. Through a further study based on the research results in [6], Kota et al. [12] derived an optimal analytical expression for retrieving a single item when one or two escorts are placed randomly in the PBS system. And they proposed a heuristic method of dealing with more than two escorts placed randomly in the system. However, the block move and the parallel move are not considered in their work.

Yalcin et al. [13] studied the single-item retrieval problem with multiple escorts placed randomly in the PBS system. They used an exact search algorithm and developed several search-guiding estimate functions to reduce the search space. To solve larger problem instances, they presented a heuristic method. However, in their research, only the single move is allowed.

Furmans et al. [14] proposed a puzzle-based storage system named GridFlow. They presented an optimal algorithm for retrieval of one item with one escort and one AGV. Alfieri et al. [15] proposed a puzzle-based storage system using a limited set of AGVs. They used a heuristic method similar to that in [6] to divide the system into several regions according to predetermined rules. The parallel move was taken account in their method. However, their method is limited in that the escorts of the system need to be initially arranged in a straight line near the IO point.

Mirzaei et al. [16] made a study of joint retrieval of multiple items in the PBS system. When more than one item needed to be retrieved, the items would be placed at a "joining location" and then moved to the IO point together. They proposed an optimal method for the joint retrieval of two items and a heuristic method for the joint retrieval of three or more items. However, in their work, it is assumed that only one escort is initially located at the IO point and the merged move is not considered.

Gue et al. [17] proposed a puzzle-based picking system named GridStore with decentralized control. The system consists of many conveyor modules. Each conveyor module

can move items in four directions. It has its own independent controller and communicates with its adjacent conveyor to decide its next action. The authors gave the proof that the system is deadlock free and studied the performance of the system in several configurations. Shekari Ashgzari and Gue [18] proposed a puzzle-based goods-to-person system named "GridPick+". The system is based on a grid of conveyor modules and decentralized control similar to the GridStore in [17]. It can bring the required items to four edges of a rectangular grid while GridStore can only bring the items to the southern boundary.

Zaerpour et al. [10] proposed a multilevel puzzle-based storage system, which is referred to as the Live-Cube system. They assumed having sufficient escorts on each level to create a virtual aisle for the requested item. A closed-form formula for expected retrieval time was derived by using traditional methods for the aisle-based system. Zaerpour et al. [19] derived the closed-form formulas for the Live-Cube system with two-storage classes. Zaerpour et al. [20] estimate the optimal layout for the Live-Cube system with two-storage classes. However, these works are based on the virtual aisle strategy, assuming that there are enough escorts on each level.

The above literature focuses on the PBS system in the form of a single PBS module or multilevel PBS system with a specific configuration. The modular system consisting of multiple PBS modules was not studied yet. Moreover, most of the above work did not allow the merged move (including both the block move and the parallel move at the same time). Although the merged move can be performed through the decentralized control and communication protocol in the GridStore proposed by Gue et al. [17], it is difficult to control each storage cell independently in other applications.

Differing from the above literature, this paper studies a modular puzzle-based storage system (MPBS system), which consists of multiple cyclic-structure PBS modules. The system has multiple IO points for transacting a batch of orders at the same time. When a batch of orders comes in advance, the MPBS system will prearrange the items near each IO point to made them consistent with each order. A path planning method considering the merged move is proposed. It involves two stages: the planning of single moves and the merging of single moves. As shown in Figure 3, in the first stage, the sequence of single moves of each PBS module is obtained without consideration of the merged move; in the second stage, the single moves in the sequence are merged into block moves and parallel moves, to reduce the steps of the system. This paper offers the following contributions:

(1) A new type of PBS system consisting of multiple PBS modules is introduced, which may motivate a further study of this new kind of PBS systems

(2) A single move planning method is proposed to solve the rearrangement problem in the MPBS system

(3) The methods of merging single moves into the block move and merging single moves into the parallel move are proposed, respectively, to reduce the steps of movement

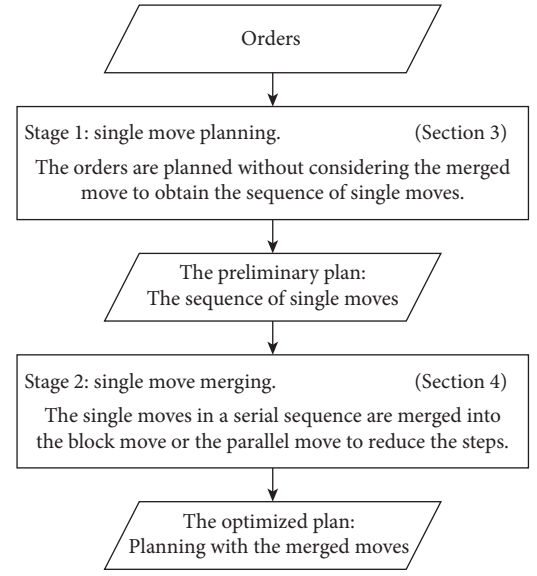


Figure 3: A two-stage path planning method considering the merged move.

(4) A planning framework that generates single move plan in the first stage and merges single moves in the second stage is proposed, for the PBS systems to allow multiple items to move simultaneously

The remainder of this paper is organized as follows. Section 2 describes the structure and workflow of the MPBS system as well as two subproblems discussed in this paper. These two subproblems correspond to the two stages mentioned above, respectively. Section 3 proposes the single move planning method aiming at the first subproblem. Section 4 refers to the methods of merging single moves into block moves and parallel moves to deal with the second subproblem. Section 5 makes an analysis of simulation and experimental results. Section 6 gives a conclusion and discusses the directions for future research.

2. Problem Description

The structured components and related concepts involved in this paper are described as follows:

Module: it means a PBS module. Multiple PBS modules can be connected to form an MPBS system.

Slot: it means the storage cell of the PBS module. A slot may be either empty or occupied by an item.

Escort: it means the empty slot.

Port: it is a slot in a module connected to adjacent module (see Figure 4).

Input/output (IO) point: the IO point is a slot in a PBS module for items to be picked out.

Picking module: it is a PBS module with an IO point to pick items out (see Figure 4).

Stocking module: it is a PBS module without an IO point, which can store items and move them to the picking module (see Figure 4).

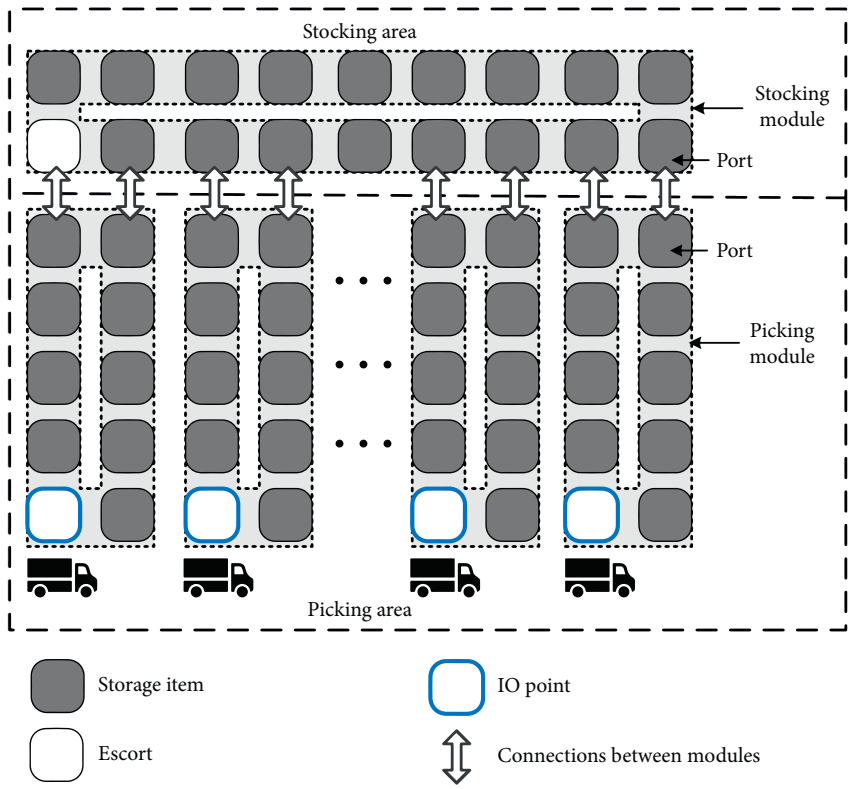


Figure 4: Schematic diagram of an MPBS system.

Move: it means a process which can be performed in one step in the MPBS system. The move includes both a single move and a merged move whose definitions are explained in Section 1 and shown in Figure 2. The single move includes both an internal move and an exchange move.

Internal move: it means moving an item inside a module.

Exchange move: it means moving an item from a port (which stores an item) of a module to another port (which is an escort) of its adjacent module, that is, the two connected modules exchange an escort and an item. An exchange move is a common move of two connected modules.

Exchange demand: if a module needs to move an item in or out, it will send an exchange demand to its adjacent module. One exchange demand corresponds to one exchange move.

State: the state of a module is described by the storage status on each slot. The state of the MPBS system involves the state of every module.

Order: the order includes a series of items that need to be picked out, according to their types and sequence. One order corresponds to one picking module.

2.1. MPBS System. As shown in Figure 4, the MPBS system is divided into a picking area and a stocking area. The stocking area has one PBS module called stocking module, and the picking area has multiple PBS modules called picking modules. Each storage module is a cyclic structure. Every picking module connected to the stocking module has an IO point in the lower left corner. Each module keeps at least one escort to guarantee the normal operation of the system. There are different types of items stored in the system.

Each picking module can be used to pick items out. The whole system can deal with a batch of orders for customers simultaneously. When a picking module receives a request to bring a certain type of items out, it will search for the required type of items within itself and then move the items to the IO point. If the picking module does not contain the required type, it will send a demand signal to the stocking module. Then, the stocking module will search for the required type and move the items to the picking module. After that, the picking module will move the required type of item to its IO point.

In practical application, a batch of orders tend to be assigned to individual picking modules in advance. In this case, the MPBS system works in a mode of preparing orders; in other words, each picking module that has received an order will arrange items near its IO point in accordance with the order. Thus, the items can be picked out in sequence for customers to get them in time.

2.2. Single Move Planning for Order Rearrangement. An order is represented by $\mathbf{Q}_k = \{q_{k,1}, \ldots, q_{k,i}, \ldots, q_{k,n_k}\}$, where i is the serial number of the orders, q_i is the type of the i-th item to be picked out, and $k \in \{1, 2, \ldots, K\}$ is the index of the picking module, where K is the total number of picking modules.

Definition 1. Order rearrangement (for a single picking module).

$\mathbf{Q}_k = \{q_{k,1}, \ldots, q_{k,i}, \ldots, q_{k,n_k}\}$ represents the order given to the picking module k. If the picking module k reaches the following states: the type of item in slot 1 is $q_{k,1}$, the type of item in slot is $q_{k,2}$, the type of item in slot n_k, is q_{k,n_k}, then it can be said that the picking module fulfill its order rearrangement.

According to Definition 1, the order $\mathbf{Q}_k$ indicates a target state of picking module k. For a batch of orders $\{\mathbf{Q}_k\}$, if each picking module that receives an order reaches its target state, then the whole MPBS system will reach target state and complete the rearrangement of that batch of orders. The single move planning for the rearrangement problem of the MPBS system is to find a sequence of single moves of each PBS module, thus to convert the MPBS system to a target state.

2.3. The Merging Problem of Single Moves. Considering the generality, it is possible to use a 4 × 4 PBS module as an example to describe the process of merging single moves. The words (*Location, Direction*) are used to indicate a move.

Location means the position where the item moves during a single move. It is represented by $l_{i,j}$, and i and j denote the column and row of the items, respectively. *Location* refers to multiple positions where items move during a block move.

Direction means the moving direction during a move. There are four possible values for the variable *Direction*: U (up), D (down), R (right), and L (left).

For example, ($l_{1,1}$, R) means that the item at position (1, 1) moves to the right. ($l_{2,1}$, $l_{1,1}$, R) means the items at position (2, 1) and (1, 1) move one step to the right at the same time during a block move.

When the merged move is not considered, the PBS module performs a sequence of single moves. In this serial sequence, some single moves that may be merged into a block move or a parallel move, so they can take place simultaneously. In the single move sequence shown in Figure 5, ($l_{2,1}$, L) and ($l_{3,1}$, L) can be merged into a block move ($l_{2,1}$, $l_{3,1}$, L), thereby reducing the system execution time from 4 steps to 3 steps. In the single move sequence shown in Figure 6, ($l_{1,2}$, U) and ($l_{3,2}$, D) can be merged into a parallel move; ($l_{1,1}$, U) and ($l_{3,1}$, L) can be merged into a parallel move, thereby reducing the system execution time from 5 steps to 3 steps.

Given a sequence of single moves, the merging problem of single moves is to merge the sequence of single moves into block moves and parallel moves, so as to reduce the steps of movement as more as possible.

3. Single Move Planning

In the PBS module, each module has its own controller, and so it can carry out the internal moves independently. However, the exchange move is performed by two connected modules. The single move planning aims at finding a sequence of moves of each module which includes internal moves and exchange moves. According to the planning, the picking module that receives an order is converted to its target state. The process of single move planning is shown in Figure 7.

3.1. The Order Matching. The order matching means getting part of items in an initial state of the picking module to match with part of items in a target state. As shown in Figure 8, the number in the middle of each slot indicates the type of item, and 0 represents an escort. The number in the lower left corner of each slot represents its serial number. Slot 1 is the IO point, and slots 4 and 5 are the ports connected to the stocking module. In Figure 8, the items at slot 1, slot 2, slot 6, and slot 7 in the current state (the corresponding item types are [1, 2, 2, 3]) and the items at slot 1, slot 2, slot 3, and slot 4 in the target state (the corresponding item types are also [1, 2, 2, 3]) are the same in type and logic sequence. Matching items in pairs in a current state and in a target state is called the matching of the picking module (current state) with its order (target state).

According to the matched group, a sequence of single moves can be generated to transform the picking module to the target state. The unmatched items in the target state can be moved into the picking module and the unmatched items in the current state can be moved out. Each moving-in or moving-out needs an exchange move, from which an inference is drawn: the less the number of matched items, the more the demand for exchange moves. It should be noted that the exchange move requires the coordination of the stocking module and the picking module. As there is only one stocking module in the MPBS system, the stocking module should deal with the demands of multiple picking modules one by one to avoid deadlock. Therefore, more exchange moves may lead to an increase of the steps for completing the rearrangement, which is unexpected.

Based on the above description, an order matching (*OM*) algorithm is introduced to get an optimal matched group with the largest number of matched items. The *OM* is described in Algorithm 1using the notation given in Table 1. Given the order $\mathbf{Q}$ and the current state S of a picking module, *OM* returns its optimal matched group. Then, i slots are from $\mathbf{Q}$ and $\mathbf{S}$ to form a subsequence p_{tar} and p_{cur}, respectively (lines 2–5). If the results of checking p_{tar} and p_{cur} are the same, then a matched group can be obtained and added to the set M (lines 6–7). In M, the matched group with the most items is the optimal result and returned (line 13).

3.2. Single Move Planning Based on Matching. The single move planning based on matching (*SMPBM*) is described in Algorithm 2 with additional notation given in Table 2. The unmatched items in the target state do not exist in the current state, so it is necessary to move them from the stocking module into the picking module(lines 3–7). The unmatched items in the current state are redundant, and it is necessary to move them out from the picking module to the stocking module (lines 8–12). For the moving-in case, the picking module moves an escort to its port to wait for the stocking module to move the required items to its

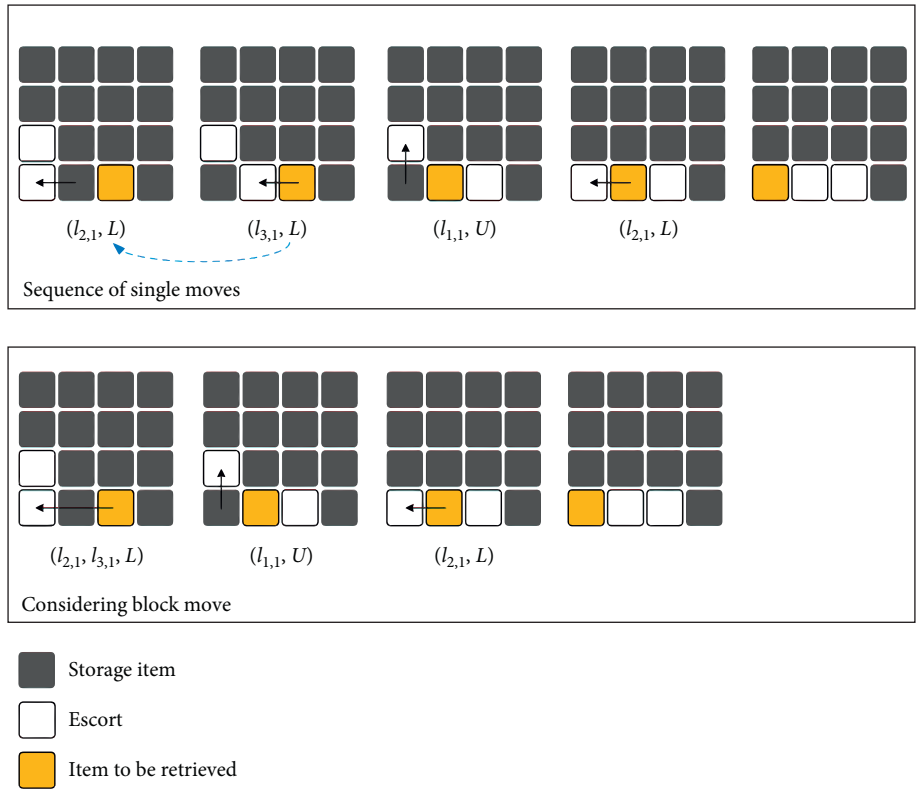


Figure 5: Diagram of the block move.

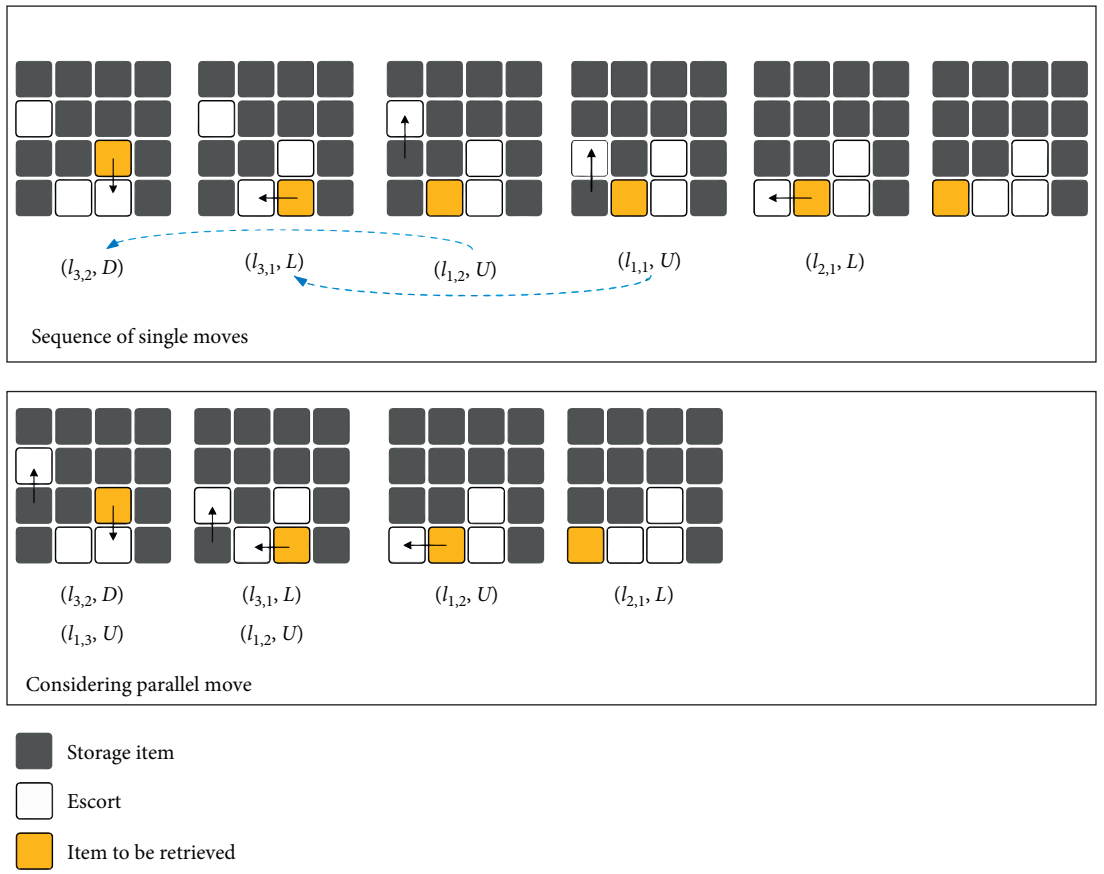


Figure 6: Diagram of the parallel move.

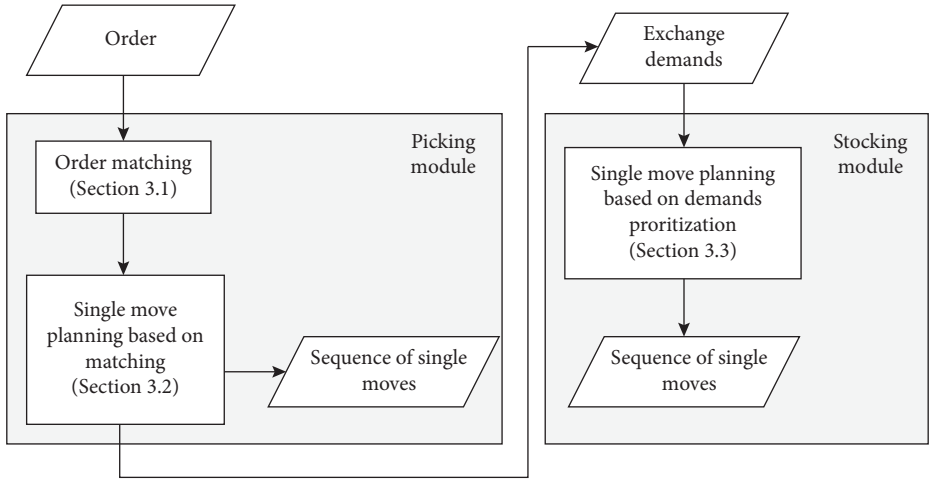


Figure 7: The process of single move planning.

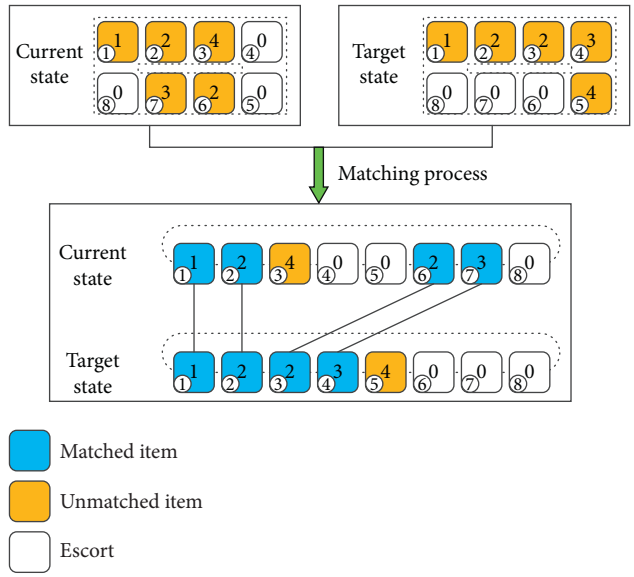


Figure 8: A case of matching process of the picking module.

corresponding port and then perform an exchange move. For the moving-out case, the picking module moves the required items to its port to wait for the stocking module to move an escort to its corresponding port and then perform an exchange move. The way to move an escort or item to the target slot inside a module is realized by the move planning of cyclic structure (*MPOCS*) algorithm (Algorithm 3). In the cyclic structure, items can only move clockwise (lines 2–13) and anticlockwise (lines 14–26). For each shift, the moving sequence of both the directions can be obtained simultaneously. Thus, it is possible to select the shorter sequence of the two as a move plan.

The process of conversion from the current state to the target state shown in Figure 9 is corresponding to the matched group in Figure 8. A single move is expressed as $\boldsymbol{A}$ = (source($\boldsymbol{A}$), direction($\boldsymbol{A}$)). Refer to the first line of Table 2 about the details of $\boldsymbol{A}$. In Figure 9, the internal move (3, 1) converts the system from the state in Figure 9(a) to the state in Figure 9(b); the exchange move (4, −2) converts the system from the state in Figure 9(b) to the state in Figure 9(c), the internal move sequence {(6, −1), (5, −1), (4, −1), (7, −1), (6, −1), (5, −1)} converts the system from the state in Figure 9(c) to the state in Figure 9(d); and the exchange move [5, 2] converts from the state in Figure 9(d) to the target state in Figure 9(e). In summary, a single move plan {(3, 1), (4, −2), (6, −1), (5, −1), (4, −1), (7, −1), (6, −1), (5, −1), (5, 2)} converts the picking module to the target state.

3.3. Single Move Planning Based on the Priority of Demands. The stocking module receives the exchange demands from each picking module and deals with them according to priority. The single move planning based on the priority of demands (*SMPBPD*) is described as Algorithm 4. The distance($\boldsymbol{d}$) denotes the remaining steps of the picking

Input: order Q, current state S
Output: optimal matched group m_{opt}.
(1) Initialize, $N = \text{length}(Q) i = N, M = \varnothing$
(2) **while** $i > 0$ **do**
(3) get $\text{comb}(\mathbf{Q}, i)$,$\text{comb}(\mathbf{S}, i)$
(4) **foreach** $p_{tar} \in \text{comb}(\mathbf{Q}, i)$ **do**
(5) **foreach** $p_{\text{cur}} \in \text{comb}(\mathbf{S}, i)$ **do**
(6) **if** check $(p_{\text{tar}}, p_{\text{cur}}) = \text{true}$ **then**
(7) $M = M \cup m(p_{\text{tar}}, p_{\text{cur}})$
(8) **end if**
(9) **end foreach**
(10) **end foreach**
(11) $i = i - 1$
(12) **end while**
(13) $m_{\text{opt}} = \text{argmax}_{m \in M} \text{length}(m)$
(14) **return** m_{opt}

Algorithm 1: $OM(Q, S)$.

Table 1: Notation.

Symbol	Definition
$\mathbf{Q} = \{q_1, q_2, \ldots, q_n\}$	An order of a picking module, indicating a target state of the picking module.
$N = \text{length}(\mathbf{Q})$	Length of the order.
$\mathbf{S} = \{s_1, \ldots, s_j, \ldots, s_J\}$	Current state of the picking module, where $s_j$is the type of item in slot j.
$\text{comb}(\mathbf{Q}, i)$	A set of subsequences composed of i serial numbers taken from Q.
$p_{\text{tar}} \in \text{comb}(\mathbf{Q}, i)$	A subsequence composed of i serial numbers taken from Q.
$\text{comb}(\mathbf{S}, i)$	A set of subsequence composed of i serial numbers taken from S
$p_{\text{cur}} \in \text{comb}(\mathbf{S}, i)$	A subsequence composed of i serial numbers selected from S
$m = (p_{\text{tar}}, p_{\text{cur}})$	A matched group determined only by p_{tar} and p_{cur}.
$\text{length}(m)$	The number of matched items, which is also the length of p_{tar} or p_{cur}.
M	A set of matched groups.

Input: order Q, current state $\boldsymbol{S}$, optimal matched group $(p_{\text{tar}}, p_{\text{cur}})$.
Output: single move plan Π, set of exchange demands $\boldsymbol{D}$.
(1) Initialize, $N = \text{length}(Q) i = 1, \Pi = \varnothing$
(2) **while** $i \leq N$ **do**
(3) **if** $i \notin p_{\text{tar}}$ **then**
(4) Item type q_i in Q is unmatched, $\text{type}(\boldsymbol{d}) = q_i$
(5) $\pi_1 = MPOCS(\boldsymbol{S}, i_{\boldsymbol{p}}, \text{next}(i_{\boldsymbol{p}}), 1)$, $\pi_2 = MPOCS(\boldsymbol{S}, i_{\boldsymbol{p}}, \text{previous}(i_{\boldsymbol{p}}), -1)$
(6) $\pi = \text{argmin}_{\pi' \in \{\pi_1, \pi_2\}} \text{length}(\pi') \mathbf{A}_{\text{ex}} = [i_p, 2]$
(7) $\Pi = \Pi \cup \pi \cup \mathbf{A}_{\text{ex}}$, $\mathbf{D} = \mathbf{D} \cup \mathbf{d}$, update p_{tar}, p_{cur}
(8) **elseif** $i \in p_{\text{tar}}$ **and** $(i + 1) \in p_{\text{tar}}$ **and** $\text{between}(\text{matched}(i), \text{matched}(i + 1)) \neq \varnothing$ **then**
(9) **foreach** $s \in \text{between}(\text{matched}(i), \text{matched}t(i + 1))$
(10) $\pi_1 = MPOCS(\boldsymbol{S}, \text{slot}(s), i_{\boldsymbol{p}}, 1)$, $\pi_2 = MPOCS(\boldsymbol{S}, \text{slot}(s), i_{\boldsymbol{p}}, -1)$
(11) $\pi = \text{argmin}_{\pi' \in \{\pi_1, \pi_2\}} \text{length}(\pi')$, $\text{distance}(\mathbf{d}) = \text{length}(\pi)$, $\mathbf{A}_{\text{ex}} = [i_p, -2]$
(12) $\Pi = \Pi \cup \pi \cup \mathbf{A}_{\text{ex}}$, $\mathbf{D} = \mathbf{D} \cup \mathbf{d}$, update p_{tar}, p_{cur}
(13) **end foreach**
(14) **end if**
(15) $i = i + 1$
(16) **end while**
(17) **return** Π , $\boldsymbol{D}$

Algorithm 2: $SMPBM$ $(Q, S, p_{\text{tar}}, p_{\text{cur}})$.

Table 2: Additional notation.

Symbol	Definition
$\boldsymbol{A} = [\text{source}(\boldsymbol{A}), \text{direction}(\boldsymbol{A})]$	A single move, where source($\boldsymbol{A}$) denotes the source slot to be moved and direction($\boldsymbol{A}$) denotes the moving direction. For an internal move, direction($\boldsymbol{A}$) = 1 or -1, which means clockwise or anticlockwise, respectively; for an exchange move, direction($\boldsymbol{A}$) = 2 or -2, which means moving-in or moving-out, respectively.
slot(s)	The slot index of item S
$\pi = \{\boldsymbol{A}_k\}$	A sequence of single moves, where $\boldsymbol{A}_k$ is a single move.
$\Pi = \{\pi_j\}$	A single move plan of the module.
previous(i)	The previous (anticlockwise adjacent) slot index of slot i.
next(i)	The next (clockwise adjacent) slot index of slot i.
matched(i)	Slot index in p_{cur} corresponding to slot i in p_{tar}.
between(i_1, i_2)	A set of items between slot i_1 and slot i_2 in a current state S
$\boldsymbol{d} = [\text{type}(\boldsymbol{d}), \text{distance}(\boldsymbol{d})]$	An exchange demand generated by the picking module, where type($\boldsymbol{d}$) denotes the type of required item, and distance($\boldsymbol{d}$) denotes the remaining steps for exchange move of $\boldsymbol{d}$.
$\boldsymbol{D} = \{\boldsymbol{d}\}$	A set of exchange demands generated by the picking module.
$i_{\boldsymbol{p}}$	The index of port slot.

Input: the current state $\boldsymbol{S}$, the source slot i_{src} of item, the destination slot i_{des} to be moved to, the moving direction *dir*.
Output: single move sequence π that moves the item at slot i_{src} to slot i_{des}.

```
(1)  Initialize, π = ∅
(2)  if dir = 1 then
(3)    i_src' = i_src
(4)    while i_src' ≠ i_des do
(5)      i_temp = next(i_src')
(6)      while type(i_temp)≠0 do
(7)        i_temp = next(i_temp)
(8)      end while
(9)      A_Inter = [previous(i_temp), 1], π = π ∪ A_inter, update S
(10)     if i_src' = previous(i_temp) then
(11)       i_src' = next(i_src')
(12)     end if
(13)   end while
(14) else
(15)   i_src' = i_src
(16)   while i_src' ≠ i_des do
(17)     i_temp = previous(i_src')
(18)     while type(i_temp)≠0 do
(19)       i_temp = previous(i_temp)
(20)     end while
(21)     A_Inter = [next(i_temp), −1], π = π ∪ A_inter, update S
(22)     if i_src' = next(i_temp) then
(23)       i_src' = previous (i_src')
(24)     end if
(25)   end while
(26) end if
(27) return π
```

Algorithm 3: *MPOCS* ($\boldsymbol{S}$, i_{src}, i_{des}, *dir*).

module for the exchange move corresponding to the demand $\boldsymbol{d}$. The smaller the distance($\boldsymbol{d}$) is, the higher the priority of demand $\boldsymbol{d}$ is. In each loop, the stocking module selects the highest priority demand to handle (line 3). The process of dealing with a demand is to move the required item or escort its corresponding port, which invokes Algorithm 3, obtaining move plans in two directions and selecting the shorter one.

4. Single Move Merging

4.1. Matrix Model of Movement. The state of a PBS module is described by a row vector. Suppose a PBS module has n slots and its state is denoted by a $1 \times n$ row vector. Then, $\mathbf{S} = [w_1, \ldots, w_i, \ldots, w_n]$, where w_i is the type of items in slot i, and $i \in \{1, 2, \ldots, n\}$. In particular, $w_i = 0$ indicates that slot i is an escort.

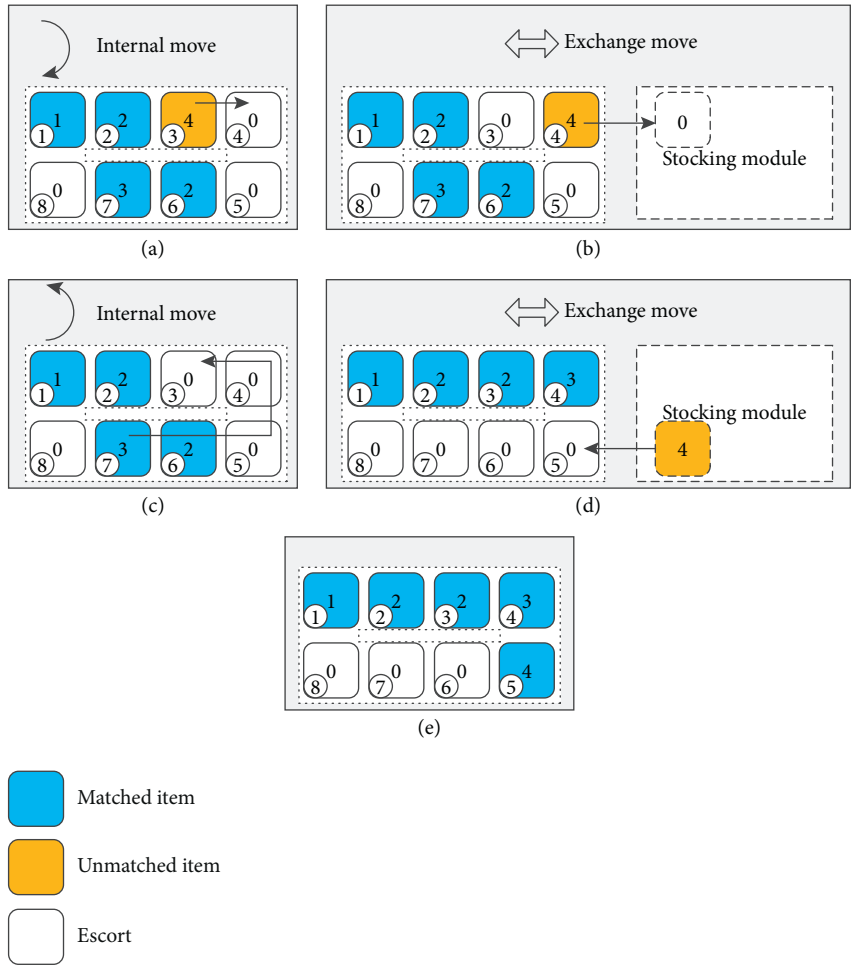


Figure 9: The process of converting a picking module to a target state.

Input: set of exchange demands $\boldsymbol{D}$. the current state S
Output: single move plan Π of stocking module.

(1) Initialize, $\Pi = \varnothing$, $\pi_s = \varnothing$
(2) **while** $\mathbf{D} \neq \varnothing$ **do**
(3) $\mathbf{d} = \text{argmin}_{d' \in \mathbf{D}} \text{distance}(d')$
(4) **if** type($\boldsymbol{d}$) = 0 **then**
(5) $\pi_1 = MPOCS(\boldsymbol{S}, i_{\boldsymbol{p}}, \text{next}(i_{\boldsymbol{p}}), 1)$, $\pi_2 = MPOCS(\boldsymbol{S}, i_{\boldsymbol{p}}, \text{previous}(i_{\boldsymbol{p}}), -1)$
(6) $\pi = \text{argmin}_{\pi' \in \{\pi_1, \pi_2\}} \text{length}(\pi')$, $\Pi = \Pi \cup \pi$
(7) **else**
(8) **foreach** $s \in \{s' | s' = \text{type}(\mathbf{d}), s' \in \mathbf{S}\}$ **do**
(9) $\pi_1 = (\boldsymbol{S}, \text{slot}(s), i_{\boldsymbol{p}}, 1)$, $\pi_2 = (\boldsymbol{S}, \text{slot}(s), i_{\boldsymbol{p}}, -1)$
(10) $\pi = \text{argmin}_{\pi' \in \{\pi_1, \pi_2\}} \text{length}(\pi')$, $\pi_s = \pi_s \cup \pi$
(11) **end foreach**
(12) $\pi = \text{argmin}_{\pi' \in \pi_s} \text{length}(\pi')$, $\Pi = \Pi \cup \pi$, $\pi_s = \varnothing$
(13) **end if**
(14) $\mathbf{D} = \mathbf{D} / \{\mathbf{d}\}$
(15) **end while**
(16) **return** Π

Algorithm 4: *SMPBPD* ($\boldsymbol{D}$, $\boldsymbol{S}$).

The movement of PBS module can be described by a matrix. Suppose the movement of a PBS module with n slots is expressed as a matrix $\mathbf{C} \in \mathrm{B}^{n\times n}$, where $\mathbf{B} = \{0, 1\}$ is the Boolean space. Then, $\mathbf{C}$ can be expressed as

$$\mathbf{C} = \left[c_{ij}\right], \quad c_{ij} \in \{0, 1\}. \tag{1}$$

In equation (1), $c_{ij} = 1$ means that the item of slot i is moved to slot j after the movement of $\boldsymbol{C}$.

Take a 2×2 PBS module for example. As shown in Figure 10, the number in the lower left corner of each slot represents the slot index of the PBS module. According to the above description, the initial state of the module can be expressed as $\mathbf{S}_0 = [w_1, w_2, 0, w_4]$. After the movement of $\boldsymbol{C}$, the item at slot 2 is moved to slot 3 while the items at slot 1 and slot 4 remain unmoved. As a result, $c_{11} = 1$, $c_{44} = 1$, and $c_{23} = 1$. Then, $\boldsymbol{C}$ can be expressed as a 4×4 matrix in equation (2). Note that $\mathbf{S}_1 = [w_1, 0, w_2, w_4]$ and $\mathbf{S}_0\mathbf{C} = [w_1, 0, w_2, w_4]$; thus, $\mathbf{S}_1 = \mathbf{S}_0\mathbf{C}$. Through the above model, the state transition of a movement can be described by the multiplication operation of a matrix and a vector:

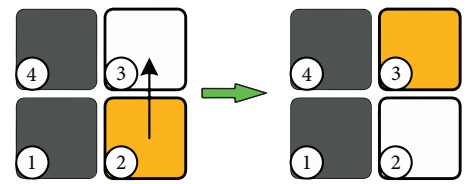
$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2}$$

4.2. Merging into Block Move. For a designed PBS module, all of its single moves and block moves can be enumerated. Therefore, all single moves can be listed as a finite set $\Theta = \{\mathbf{C}_i\}$. Furthermore, all single move that can be merged into block moves can be enumerated.

Definition 2. The longest block move.

The longest block move is a sequential set of all single moves which can be merged into a block move on a straight path (row or column) of PBS module. It is expressed as $\mathbf{C}_{\mathrm{LB}} = \{\mathbf{C}_1, \ldots, \mathbf{C}_i, \ldots, \mathbf{C}_n\}$. For all straight paths of the PBS module, their corresponding longest block moves can be enumerated as a finite set $\mathbf{G} = \{\mathbf{C}_{\mathrm{LB},1}, \mathbf{C}_{\mathrm{LB},2}, \ldots, \mathbf{C}_{\mathrm{LB},X}\}$.

The block move merging (*BMM*) algorithm is described in Algorithm 5. Given a sequence of single moves $\Pi = \mathbf{C}_1, \ldots, \mathbf{C}_i, \ldots, \mathbf{C}_m$ to be performed and the set of all longest block moves $\boldsymbol{G}$, *BMM* returns a block move that can be performed at the current step. $\mathbf{C}_1$ is the first move to be performed at the current step, and its subsequent move sequence is $\Pi_s = \mathbf{C}_2, \ldots, \mathbf{C}_i, \ldots, \mathbf{C}_m$. By traversing the set $\boldsymbol{G}$, the longest block move $\mathbf{C}_{\mathrm{LB},x}$ that contains $\boldsymbol{C}_1$ (lines 2–4) will be found. Suppose $\mathbf{C}_{\mathrm{LB},x,i} = \mathbf{C}_1$ and $\mathbf{C}_{\mathrm{LB},x,i+j}$ is the j-th move after $\mathbf{C}_{\mathrm{LB},x,i}$ in $\mathbf{C}_{\mathrm{LB},x}$, where $\mathbf{C}_{\mathrm{LB},x,i} \in \mathbf{C}_{\mathrm{LB},x}$ and $\mathbf{C}_{\mathrm{LB},x,i+j} \in \mathbf{C}_{\mathrm{LB},x}$. Let $\mathbf{C}_{1+j}$ denote the j-th move after $\mathbf{C}_1$ in Π. In Algorithm 5, j is initialized to 1 (line 1) and incremented in each loop. According to Definition 2, the continuous moves in $\mathbf{C}_{\mathrm{LB},x}$ can be merged into a block move. Judging whether $\mathbf{C}_{1+j} = \mathbf{C}_{\mathrm{LB},x,i+j}$ in each loop, then the block move $\mathbf{C}_{\mathrm{block}}$ at the current step can be obtained (lines 5–6).



The state before moving: S_0 The state after moving: S_1

Figure 10: The process of state transition during a movement.

4.3. Merging into Parallel Move. According to Section 4.1, if a sequence of moves $\mathbf{C}_1, \ldots, \mathbf{C}_m$ converts the PBS module from the initial state $\mathbf{S}_{\mathrm{init}}$ to the state $\mathbf{S}_m$, then this process is expressed as

$$\mathbf{S}_m = \mathbf{S}_{\mathrm{init}}\mathbf{C}_1, \ldots, \mathbf{C}_m. \tag{3}$$

In a sequence of moves $\mathbf{C}_1, \ldots, \mathbf{C}_m$, $\mathbf{C}_1$ is the first move to be executed at the current step. However, in the remaining sequence of moves $\mathbf{C}_2, \ldots, \mathbf{C}_m$, there may be moves that can be executed in parallel with $\mathbf{C}_1$.

Lemma 1. *For the two single moves $\mathbf{C}_1$ and $\mathbf{C}_2$ of a PBS system, if $\mathbf{C}_2\mathbf{C}_1 = \mathbf{C}_1\mathbf{C}_2$, then $\mathbf{C}_1$ and $\mathbf{C}_2$ can be parallelly executed.*

Proof. suppose the initial state of the PBS system is $\mathbf{S}_{\mathrm{init}}$. Since that $\mathbf{C}_1\mathbf{C}_2 = \mathbf{C}_2\mathbf{C}_1$, the following equation can be derived:

$$\mathbf{S}_{\mathrm{init}}\mathbf{C}_1\mathbf{C}_2 = \mathbf{S}_{\mathrm{init}}\mathbf{C}_2\mathbf{C}_1, \tag{4}$$

which shows that the executive sequence of $\mathbf{C}_1$ and $\mathbf{C}_2$ does not affect the final state of the PBS system; that is, $\mathbf{C}_1$ and $\mathbf{C}_2$can be parallelly executed. □

Theorem 1. *For a given sequence of single moves $\mathbf{C}_1, \ldots, \mathbf{C}_i, \ldots, \mathbf{C}_m$, $\mathbf{C}_1$ is the first move to be executed at the current step and $\mathbf{C}_i$ is a certain move in the sequence to be executed later. A sufficient condition for $\mathbf{C}_i$ and $\mathbf{C}_1$ to be executed parallelly is*

$$\mathbf{C}_1, \ldots, \mathbf{C}_{i-1}\mathbf{C}_i = \mathbf{C}_i\mathbf{C}_1, \ldots, \mathbf{C}_{i-1}. \tag{5}$$

Proof. let $C' = \mathbf{C}_1, \ldots, \mathbf{C}_{i-1}$, then the formula $C'\mathbf{C}_i = \mathbf{C}_iC'$ can be derived. And the move sequence $\mathbf{C}_1, \ldots, \mathbf{C}_{i-1}$ is considered as a whole move C'. According to Lemma 1, $\mathbf{C}$' and $\mathbf{C}_i$ can be executed parallelly. Because $\mathbf{C}_1$ is the first move in C', so $\mathbf{C}_1$ and $\mathbf{C}_i$ can be executed parallelly. □

Theorem 2. *Given a sequence of single moves $\mathbf{C}_1, \ldots, \mathbf{C}_i, \ldots, \mathbf{C}_m$, it follows that $\mathbf{C}_i$,$\mathbf{C}_j$, and $\mathbf{C}_1$ can be executed parallelly if equations (6) and (7) are satisfied:*

$$\mathbf{C}_1, \ldots, \mathbf{C}_{i-1}\mathbf{C}_i = \mathbf{C}_i\mathbf{C}_1, \ldots, \mathbf{C}_{i-1}, \tag{6}$$

$$\mathbf{C}_1, \ldots, \mathbf{C}_{j-1}\mathbf{C}_j = \mathbf{C}_j\mathbf{C}_1, \ldots, \mathbf{C}_{j-1}. \tag{7}$$

Input: sequence of single moves $\Pi = \mathbf{C}_1, \ldots, \mathbf{C}_m$, a set of all the longest block moves $\boldsymbol{G}$.
Output: the block move $\mathbf{C}_{\text{block}}$ at the current step.

```
(1)  Initialize, x = 1 i = 1 j = 1, C_block = {C_1}
(2)  foreach C_LB,x ∈ G do
(3)    foreach C_LB,x,i ∈ C_LB,x do
(4)      if C_1 = C_LB,x,i then
(5)        while C_1+j = C_LB,x,i+j do
(6)          C_block = C_block ∪ C_1+j, j = j + 1
(7)        end while
(8)      end if
(9)    end foreach
(10) end foreach
(11) return C_block
```

Algorithm 5: *BMM* (Π, $\boldsymbol{G}$).

Input: sequence of single moves $\Pi = \mathbf{C}_1, \ldots, \mathbf{C}_i, \ldots, \mathbf{C}_m$.
Output: the parallel move $\mathbf{C}_{\text{para}}$ at current step.

```
(1)  Initialize, i = 2, C_para = {C_1}
(2)  while i < m do
(3)    if C_1, ..., C_i−1 C_i = C_i C_1, ..., C_i−1 then
(4)      C_para = C_para ∪ C_i
(5)    end if
(6)  end while
(7)  return C_para
```

Algorithm 6: *PMM*. ($\Pi = \mathbf{C}_1, \ldots, \mathbf{C}_i, \ldots, \mathbf{C}_m$).

Proof:. according to Theorem 1, $\mathbf{C}_i$ and $\mathbf{C}_1$ are parallel according to (6); $\mathbf{C}_j$ and $\mathbf{C}_1$ are in parallel according to (7). Assuming that $i < j$, equation (7) can be written as (8). The substitution of (6) into the left and the right side of (8) leads to (9) and (10), respectively. Thus, the right side of (9) equals to the right side of (10), which is (11). According to (11) and Theorem 1, $\mathbf{C}_i$ and $\mathbf{C}_j$ can be executed in parallel. Therefore, $\mathbf{C}_i$,$\mathbf{C}_j$, and $\mathbf{C}_1$ can be executed in parallel:

$$\mathbf{C}_1, \ldots, \mathbf{C}_{i-1}\mathbf{C}_i\mathbf{C}_{i+1}, \ldots, \mathbf{C}_{j-1}\mathbf{C}_j = \mathbf{C}_j\mathbf{C}_1, \ldots, \mathbf{C}_{i-1}\mathbf{C}_i\mathbf{C}_{i+1}, \ldots, \mathbf{C}_{j-1}, \tag{8}$$

$$\mathbf{C}_1, \ldots, \mathbf{C}_{i-1}\mathbf{C}_i\mathbf{C}_{i+1}, \ldots, \mathbf{C}_{j-1}\mathbf{C}_j = \mathbf{C}_i\mathbf{C}_1, \ldots, \mathbf{C}_{i-1}\mathbf{C}_{i+1}, \ldots, \mathbf{C}_{j-1}\mathbf{C}_j, \tag{9}$$

$$\mathbf{C}_j\mathbf{C}_1, \ldots, \mathbf{C}_{i-1}\mathbf{C}_i\mathbf{C}_{i+1}, \ldots, \mathbf{C}_{j-1} = \mathbf{C}_j\mathbf{C}_i\mathbf{C}_1, \ldots, \mathbf{C}_{i-1}\mathbf{C}_{i+1}, \ldots, \mathbf{C}_{j-1}, \tag{10}$$

$$\mathbf{C}_i\mathbf{C}_1, \ldots, \mathbf{C}_{i-1}\mathbf{C}_{i+1}, \ldots, \mathbf{C}_{j-1}\mathbf{C}_j = \mathbf{C}_j\mathbf{C}_i\mathbf{C}_1, \ldots, \mathbf{C}_{i-1}\mathbf{C}_{i+1}, \ldots, \mathbf{C}_{j-1}. \tag{11}$$

A sequence of single moves $\Pi = \mathbf{C}_1, \ldots, \mathbf{C}_i \ldots, \mathbf{C}_m$ is given, according to Theorem 2, if multiple single moves satisfy the sufficient condition of equation (5) in Theorem 1, then these moves are all parallel with one another. Based on the above principle, a parallel move merging (*PMM*) algorithm (Algorithm 6) is proposed. $\mathbf{C}_1$ is the first move to be executed at the current step and its subsequent move sequence is $\Pi_s = \mathbf{C}_2, \ldots, \mathbf{C}_i, \ldots, \mathbf{C}_m$. The move $\mathbf{C}_i$ is taken out from $\Pi_s = \mathbf{C}_2, \ldots, \mathbf{C}_i, \ldots, \mathbf{C}_m$ to judge whether $\mathbf{C}_i$ and $\mathbf{C}_1$ satisfy equation (5) in each loop (lines 1–4). If $\mathbf{C}_i$ and $\mathbf{C}_1$

satisfy equation (5), $\mathbf{C}_i$ will be added to $\mathbf{C}_{\text{para}}$ (line 4). Finally, all parallel moves at the current step can be obtained (line 7). □

5. Simulation and Analysis

5.1. Simulation Settings. The MPBS system used for simulation consists of one stocking module and ten picking modules:

(i) Fixed parameters: 50 slots of the stocking module, 10 slots of each picking module, with its slot 5 and slot 6 served as ports and connected with the stocking module, and four types of items stored in the whole system.

(ii) Variable parameters: the number of orders α, which represents the number of orders assigned to the system, or the number of picking modules needing to be rearranged; the storage density ρ, which represents the ratio of the number of items to the total number of slots in a PBS module.

The simulations on the number of orders α and the storage density ρ are carried out, respectively. The given parameters are used to generate a series of random problem instances. For each instance, the order assigned to a picking module is generated by randomly disrupting the initial state of this picking module.

The simulation results of each instance are divided into 4 cases: the results using only single move planning (denoted by *Single*); the results using single move planning and block move merging (denoted by *Block*); the results using single move planning and parallel move merging (denoted by *Para*); the results using single move planning, both block move merging and parallel move merging (denoted by *Both*). The average steps of the above 4 cases are separately defined as (12)–(15), where T is the number of instances under each parameter configuration:

$$\text{avg}_{\text{Single}} = \frac{1}{T}\sum_{i=1}^{T} \text{Single}_i, \tag{12}$$

$$\text{avg}_{\text{Block}} = \frac{1}{T}\sum_{i=1}^{T} \text{Block}_i, \tag{13}$$

$$\text{avg}_{\text{Para}} = \frac{1}{T}\sum_{i=1}^{T} \text{Para}_i, \tag{14}$$

$$\text{avg}_{\text{Both}} = \frac{1}{T}\sum_{i=1}^{T} \text{Both}_i. \tag{15}$$

Furthermore, the average optimization ratios of the latter three cases (*Block*, *Para*, and *Both*) relative to the single case are, respectively, defined as (16)–(18), where T is the number of instances under each parameter configuration:

$$\eta_{\text{Block}} = \frac{1}{T}\sum_{i=1}^{T} \frac{\text{Single}_i - \text{Block}_i}{\text{Single}_i} \times 100\%, \tag{16}$$

$$\eta_{\text{Para}} = \frac{1}{T}\sum_{i=1}^{T} \frac{\text{Single}_i - \text{Para}_i}{\text{Single}_i} \times 100\%, \tag{17}$$

$$\eta_{\text{Both}} = \frac{1}{T}\sum_{i=1}^{T} \frac{\text{Single}_i - \text{Both}_i}{\text{Single}_i} \times 100\%. \tag{18}$$

5.2. Experimental Results concerning the Number of Orders. The storage density is set as 80%, and the length of each order is set as 8. The value of the number of orders α is changed, and 50 instances are generated for each value of α. Table 3 gives the numerical results, and Figure 11 shows the histogram.

From Table 3 and Figure 11, it can be seen that all the four methods (*Single*, *Block*, *Para*, and *Both*) can solve the rearrangement problem stably in different number of orders. With an increase in the number of orders, the average number of steps increases, which is approximately linear with the number of orders. The number of orders is also the number of picking modules that need to be rearranged. For each picking module, its average number of steps and average number of exchange demands tend to be similar with many random experiments. For the stocking module, many random experiments show that its average number of steps is approximately linear to the number of exchange demands. Therefore, the average number of steps increases approximately linearly with the number of orders.

The relationship among the average steps of the four methods is $\text{avg}_{\text{Both}} < \text{avg}_{\text{Block}} < \text{avg}_{\text{Para}} < \text{avg}_{\text{Single}}$, and the relationship between the average optimization ratios is $\eta_{\text{Para}} < \eta_{\text{Block}} < \eta_{\text{Both}}$. This is because the storage density of each module is set as 0.8. Thus, there are fewer escorts in the system but more items can be moved as a block. In this case, there are fewer moves that can be merged into parallel moves, while there are more moves that can be merged into block moves.

In addition, the average optimization ratio of case *Both* is over 60% as far as the orders different in number are concerned. It indicates that the merged moves (including both block move and parallel move) can greatly reduce the steps and improve the execution efficiency.

5.3. Experimental Results concerning the Storage Density. The number of orders is set as 8. The value of storage density ρ is changed, and 50 instances are generated for each value of ρ. Table 4 gives the numerical results, and Figure 12 shows the histogram. Two simulation videos are provided in Supplementary Materials.

According to Table 4 and Figure 12, all the four methods (*Single*, *Block*, *Para*, and *Both*) can be used to solve the rearrangement problem stably in different storage densities. With an increase in storage density, the average number of

Table 3: Experimental results of different number of orders.

Parameter configurations		Results of case *Single*	Results of case *Block*		Results of case *Para*		Results of case *Both*	
ρ	α	$\text{avg}_{\text{Single}}$	$\text{avg}_{\text{Block}}$	η_{Block}	avg_{Para}	η_{Para}	avg_{Both}	η_{Both}
80%	6	408.58	146.94	0.6404	255.20	0.3754	128.44	0.6856
80%	7	423.00	161.16	0.6190	295.96	0.3003	145.92	0.6550
80%	8	486.62	173.98	0.6425	304.58	0.3741	154.78	0.6819
80%	9	587.46	196.80	0.6650	377.18	0.3579	178.48	0.6962
80%	10	632.62	213.90	0.6619	398.84	0.3695	194.02	0.6933

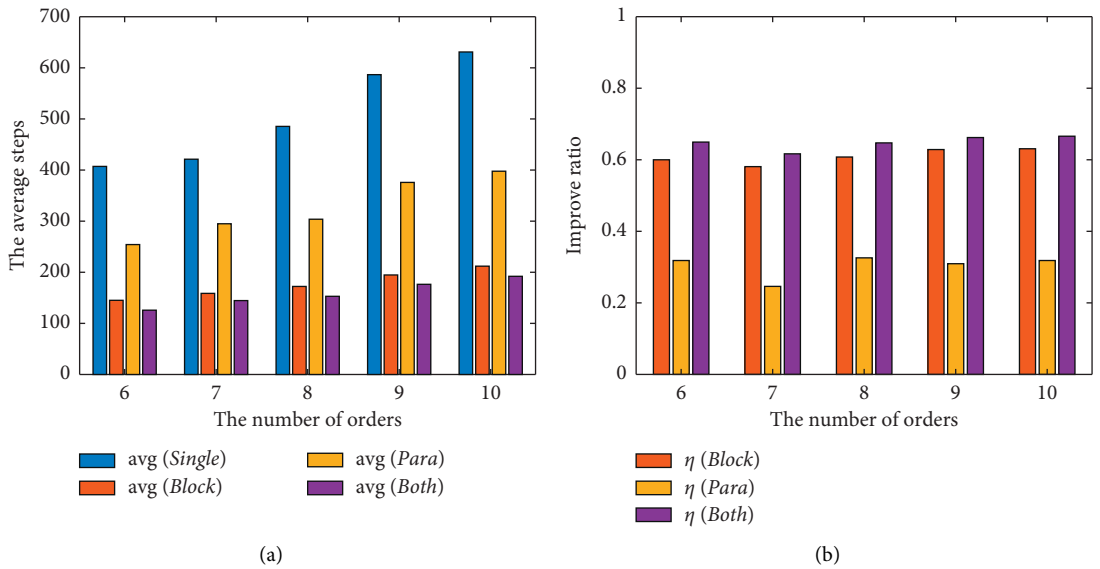


Figure 11: Comparison of four cases according to different number of orders. (a) The average steps. (b) The average optimization ratio.

Table 4: Experimental results of different storage densities.

Parameter configurations		Results of case *Single*	Results of case *Block*		Results of case *Para*		Results of case *Both*	
ρ	α	$\text{avg}_{\text{Single}}$	$\text{avg}_{\text{Block}}$	η_{Block}	avg_{Para}	η_{Para}	avg_{Both}	η_{Both}
50%	8	171.78	102.94	0.4007	87.68	0.4896	81.40	0.5261
60%	8	241.22	144.42	0.4013	141.18	0.4147	119.68	0.5039
70%	8	413.30	206.66	0.5000	216.58	0.4760	167.26	0.5953
80%	8	608.48	244.32	0.5985	391.66	0.3563	214.30	0.6478
90%	8	948.38	310.06	0.6731	756.26	0.2026	275.96	0.7090

steps increases, and the increasing speed becomes faster and faster. The reason is that, on one hand, the increase of storage density leads to the decrease of escorts in the system, resulting in that there are more obstructive items on the moving path need to be cleared; on the other hand, the length of each order increases with the storage density, which will cause more difficulty for rearrangement.

When the storage density is low, the relationship between the average steps of the four methods is $\text{avg}_{\text{Both}} < \text{avg}_{\text{Para}} < \text{avg}_{\text{Block}} < \text{avg}_{\text{Single}}$, and the relationship between the average optimization ratios is $\eta_{\text{Block}} < \eta_{\text{Para}} < \eta_{\text{Both}}$. With an increase in storage density, the average steps increase, but $\text{avg}_{\text{Single}}$ and avg_{Para} increase faster than $\text{avg}_{\text{Block}}$ and avg_{Both}; η_{Block} decreases, but η_{Para} and η_{Both} increase. When the storage density exceeds 0.7, the relationship between the average steps becomes $\text{avg}_{\text{Both}} < \text{avg}_{\text{Block}} < \text{avg}_{\text{Para}} < \text{avg}_{\text{Single}}$, and the relationship between the average optimization ratios becomes $\eta_{\text{Para}} < \eta_{\text{Block}} < \eta_{\text{Both}}$. The above findings show that, with an increase in storage density, the number of stored items increases while the number of escorts decreases. Thus, the optimization effect of the block move becomes stronger and the optimization effect of the parallel move becomes weaker.

In addition, the average optimization ratio of case *Both* is over 50% and can reach 70% when the storage density is increased to 0.9. This indicates that considering the merged

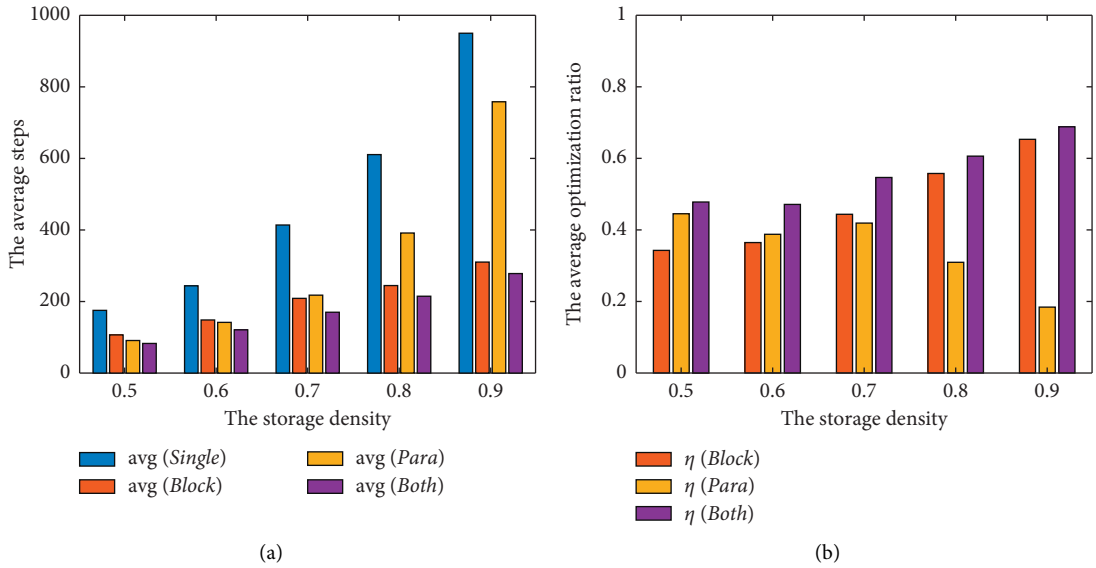


Figure 12: Comparison of four cases according to different storage densities. (a) The average steps. (b) The average optimization ratio.

moves (including both block move and parallel move) can greatly reduce the steps and improve the execution efficiency, and that the optimization effect is more significant under the condition of high storage density.

6. Conclusions

This paper introduces an MPBS system consisting of multiple cyclic PBS modules and proposes a two-stage path planning method considering the merged moves to solve the rearrangement problem. The proposed method includes single move planning and single move merging. In the stage of single move planning, a path planning method based on the order matching and the priority of demands is proposed; in the stage of single move merging, the method of merging single moves into block moves and the method of merging single moves into parallel moves are proposed, respectively. Through the simulation and analysis in different numbers of orders and different storage densities, the effectiveness of the single move planning method and the optimization performance of single move merging method are verified.

The framework of solving problems in stages may provide references for other PBS systems. For example, in the studies of the other PBS systems, there may be different methods to get a single move plan. Then, the single move plan can be merged in the second stage. The proposed method in this paper can deal with one sequence of moves, and further research may focus on the method to merge multiple sequences of moves or the planning method in stages for the other PBS systems.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research was funded by the National Natural Science Foundation of China, grant no. 52077218.

Supplementary Materials

Two simulation videos with an explanatory document can be obtained from https://drive.Google.com/drive/folders/1ToMJ-NyTDCHcEM6bCmBBrqnMEL9y_VauX?usp = sharing. (*Supplementary Materials*)

References

[1] B. Nils, R. D. Koster, and W. Felix, "Warehousing in the e-commerce era: a survey," *European Journal of Operational Research*, vol. 277, pp. 396–411, 2019.

[2] S. Zhang, L. Fu, R. Chen, and M. Yu, "Optimizing the cargo location assignment of retail e-Commerce based on an artificial fish swarm algorithm," *Mathematical Problems in Engineering*, vol. 2020, no. 5, 14 pages, Article ID 5609873, 2020.

[3] Y. A. Bozer and J. A. White, "Travel time models for automated storage/retrieval systems,," *IIE Transactions*, vol. 16, no. 4, pp. 329–338, 1982.

[4] T. Lerher, "Travel time model for double-deep shuttle-based storage and retrieval systems," *International Journal of Production Research*, vol. 54, no. 9-10, pp. 1–22, 2016.

[5] K. J. Roodbergen and I. F. A. Vis, "A survey of literature on automated storage and retrieval systems," *European Journal of Operational Research*, vol. 194, no. 2, pp. 343–362, 2009.

[6] K. R. Gue and B. S. Kim, "Puzzle-based storage systems," *Naval Research Logistics*, vol. 54, no. 5, pp. 556–567, 2007.

[7] K. R. Gue and O. Ulndag, "A high-density, puzzle-based order picking system," in *Proceedings of the 12th IMHRC*, Gardanne, France, June 2012.
[8] K. R. Gue and G. Hao, "A high-density, puzzle-based system for rail-rail container transfers," in *Proceedings of the 14th IMHRC*, Karlsruhe, Germany, June 2016.
[9] A. Yalcin, A. Koberstein, and K.-O. Schocke, "Motion and layout planning in a grid-based early baggage storage system," *OR Spectrum*, vol. 41, no. 3, pp. 683–725, 2019.
[10] N. Zaerpour, Y. Yu, and R. De Koster, "Small is beautiful: a Framework for evaluating and optimizing live-cube compact storage systems," *Transportation Science*, vol. 51, no. 1, pp. 34–51, 2017.
[11] V. R. Kota, D. Taylor, and K. R. Gue, "Retrieval time performance in puzzle-based storage systems," in *Proceedings of the IERC*, pp. 1558–1563, Cancun, Mexico, May 2010.
[12] V. R. Kota, D. Taylor, and K. R. Gue, "Retrieval time performance in puzzle-based storage systems," *Journal of Manufacturing Technology Management*, vol. 26, no. 4, pp. 582–602, 2015.
[13] A. Yalcin, A. Koberstein, and K. O. Schocke, "An optimal and a heuristic algorithm for the single-item retrieval problem in puzzle-based storage systems with multiple escorts," *International Journal of Production Research*, vol. 57, pp. 1–23, 2019.
[14] K. Furmans, C. Nobbe, and M. Schwab, "Future of material handling-modular, flexible and effcient," in *Proceedings of International Conference on Intelligent Robots and Systems*, San Francisco, CA, USA, October2011.
[15] A. Alfieri, M. Cantamessa, A. Monchiero, and F. Montagna, "Heuristics for puzzle-based storage systems driven by a limited set of automated guided vehicles," *Journal of Intelligent Manufacturing*, vol. 23, no. 5, pp. 1695–1705, 2012.
[16] M. Mirzaei, R. De Koster, and N. Zaerpour, "Modelling item retrievals in puzzle-based storage systems," *International Journal of Production Research*, vol. 55, pp. 1–13, 2017.
[17] K. R. Gue, K. Furmans, Z. Seibold, and O. Uludag, "Gridstore: a puzzle-based storage system with decentralized control," *IEEE Transactions on Automation Science and Engineering*, vol. 11, no. 2, pp. 429–438, 2014.
[18] M. Shekari Ashgzari and K. R. Gue, "A puzzle-based material handling system for order picking," *International Transactions in Operational Research*, vol. 28, no. 4, pp. 1821–1846, 2021.
[19] N. Zaerpour, Y. Yu, and R. B. M. de Koster, "Response time analysis of a live-cube compact storage system with two storage classes," *IISE Transactions*, vol. 49, no. 5, pp. 461–480, 2017.
[20] N. Zaerpour, Y. Yu, and R. B. M. de Koster, "Optimal two-class-based storage in a live-cube compact storage system," *IISE Transactions*, vol. 49, no. 7, pp. 653–668, 2017.