An efficient heuristic for minimizing the number of moves for the retrieval of a single item in a puzzle-based storage system with multiple escorts

Yunfeng Ma, Haoxun Chen, Yugang Yu

Version of Record: https://www.sciencedirect.com/science/article/pii/S0377221721008079
Manuscript_ad1c766a9cd5170c88a99958a854bb68

An efficient heuristic for minimizing the number of moves for the retrieval of a single item in a puzzle-based storage system with multiple escorts

Yunfeng MA[a,b], Haoxun CHEN[b,*], Yugang YU[c,d,*]

a School of Management, Wuhan University of Science and technology, Wuhan 430065, China

b Industrial Systems Optimization Laboratory, University of Technology of Troyes, 12 rue Marie Curie, CS 42060, 10004 Troyes, France

c International Institute of Finance, School of Management, University of Science and Technology of China, 230026 Hefei, China

d Anhui Province Key Laboratory of Contemporary Logistics and Supply Chain, School of Management, University of Science and Technology of China, 230026 Hefei, China

Abstract

The Puzzle-Based Storage (PBS) system is an innovative high-density storage system for physical goods, which has the advantage of gravity flow racks in terms of space efficiency and a relatively high accessibility to each unit-load of a variety of personalized goods stored. For the PBS system, among many intricate scheduling problems studied, the minimization of total number of item-moves when retrieving a single item with multiple escorts is an essential building block for its operation. To tackle this problem, we propose a hybrid approach that combines state appraisal, neighborhood search, and beam search. Our numerical experiments on a large number of benchmark instances show that, compared with the results of these instances provided by the best existing heuristic with computational complexity of $O(n^4)$, our algorithm with different computational complexity settings can improve the overall average solution accuracy from 1.096% to 0.055% by its setting of $O(n^3)$, or to 0.570% by its setting of $O(n)$, where n is the size of the PBS system. For PBS systems that are more in line with the actual storage density, our algorithm shows stronger robustness by improving the accuracy from 1.086% to 0.026% by its setting of $O(n^3)$, or to 0.249% by its setting of $O(n)$. The significant improvement in efficiency and accuracy of our algorithm for this basic problem makes its industrial applications in PBS systems promising.

Keywords: Heuristics, Automated warehouses, Compact storage systems, Puzzle-based storage, Beam search

1. Introduction

The development of today's e-commerce and the challenge of mass customization have significantly increased

* Corresponding authors
E-mail addresses: mayunfeng@wust.edu.cn (Yunfeng MA), haoxun.chen@utt.fr (Haoxun Chen), ygyu@ustc.edu.cn (Yugang YU).

the variety of goods and number of orders. In addition, the scarcity of urban land requires modern storage systems to be more space-efficient than ever. These situations give rise to an increasing demand on modern storage systems with compactness, flexibility, and rapidly response capability. Therefore, with very narrower aisles and high degree of automation, shuttle-based storage and retrieval systems (SBS/RS) have been widely studied and adopted in real-life applications in recent years (Küçükyaşar et al., 2021; Lerher et al., 2020; Lerher et al., 2017; Carlo et al., 2012; Marchet et al.). During almost the same period, a more compact and flexible automated storage system has gradually turned into reality, that is, the Puzzle-based storage (PBS) system. The PBS system, originally proposed by Gue and Kim (2007), is a new generation of intelligent storage system emerging in recent years. The PBS system can be regarded as one type of SBS/RS with the highest space efficiency, as well as the most complicated one in terms of shuttle/AGV scheduling. Goods can be stored in PBS system extremely compact to the limit with only one empty location (referred to as **escort**) required to realize the storage and retrieval operation, and aisles are not necessarily used. Azadeh, De Koster and Roy (2019) systematically reviewed some new types of automated and robotic warehouses. Among them, PBS systems, also called grid-based shuttle systems, are identified as one of the most promising storage systems, which need more research attentions.

A PBS system is composed of: (1) storage units (referred to as items) that can operate independently, such as AGV (Automated Guided Vehicles), shuttles, and conveyor belt modules; (2) one or multiple empty locations (escort□); (3) one or multiple depots (Input/output location or I/O location for short). An item can move in four cardinal directions, that is up, down, left, and right, respectively if the corresponding adjacent location is an escort. An item move is a single move of one item from its current location to its adjacent escort location. A series of item moves is required for retrieving a requested item (R)to an I/O location in a PBS system. Usually, for the basic problem that does not allow **block move** (simultaneous movement of multi-items in a line), the objective of a PBS scheduling problem is to minimize the total number of item-moves or retrieval time, the two objectives are identical.

The hardware and information technology related to PBS systems are mature, such as modern electronic information systems, sensors and control units, battery technology, equipment manufacturing, etc., The evolution of AGVs into autonomous mobile robots has become possible due to new hardware and software technologies (Fragapane et al., 2021). Meanwhile, the hardware costs for manufacturing robots, shuttles, and other related equipment have dropped to an acceptable range, making it possible to deploy PBS physical systems in a large variety of commercial applications. There have been some real-world implementations of

PBS in warehousing, carparking systems (Mayer, 2009; Mutrade, 2014; RR Parkon, 2014; ODTH, 2015; IFL, 2015; Doppler, 2016; Gebhard, 2016; Woehr, 2016; ICAM, 2019).

Compared to traditional warehouses with aisles, two major drawbacks of PBS systems are, respectively, longer retrieval time and too many movements of stored items. Luckily, these drawbacks can be largely overcome by properly increasing the number of escorts to make a good tradeoff among space efficiency, retrieval time and energy consumption. However, the introduction of multiple escorts into PBS systems also involves some intricate scheduling problems, the most critical one is the Single-item Retrieval Problem in PBS systems with Multiple Escorts (SRPME). Fortunately, two breakthroughs have been made related to this problem. The first one is the optimal algorithm for minimizing retrieval time, or item move time, of SRPME (T-SRPME) designed by Yu et al. (2019) by allowing block move, the second one is the optimal and heuristic algorithms for minimizing the number of item-moves of SRPME (M-SRPME) proposed by Yalcin et al. (2019a). Their works make practical operations of PBS systems feasible.

If block move is not allowed, the problem of minimizing retrieval time and that of minimizing the number of item moves are equivalent. In other words, T-SRPME is a special case of M-SRPME, where the latter is also named as SPRME in the paper of Yalcin et al. (2019a). We give different names to the two problems here to distinguish our work from that of Yu et al. (2019). In fact, M-SRPME is based on puzzle, hence a NP-hard problem. The puzzle is so hard that even very small cases, i.e., 8-puzzle (3×3) and 15-puzzle (4×4), are hard enough to be taken as workbench laboratory, for some decades, for testing the performance of searching methods (Ratner and Warmuth, 1986).

As a basic building block for designing and operating PBS systems in today's e-commerce and mass customization environment, M-SRPME is playing a vital role in PBS system design and business model implementation, however, the need for finding faster and more precise heuristics to solve it has still not been met yet. For example, Amazon's KIVA system, JD's warehousing robot handling system, and other similar SBS/RS or AGV systems can be deployed as PBS systems to improve space utilization, item accessibility, layout and control flexibility, as well as to reduce storage and retrieval time. However, the scale of these systems, if rearranged as PBS systems, would be much larger than that the existing algorithms can efficiently deal with. Although the heuristic algorithm proposed by Yalcin et al. (2019a) is good enough for the operations of small and medium-sized PBS systems, M-SRPME deserves a better algorithm in terms of both solution accuracy and computation time especially for large size PBS systems.

In this paper, we propose a hybrid algorithm that combines state appraisal, neighborhood search, and beam

search for M-SRPME. Our algorithm greatly improves the solution accuracy and computation speed compared with the best benchmark algorithm proposed by Yalcin et al. (2019a), it can therefore better support the design of larger and more complex PBS systems for fully realizing their commercial functions.

The remainder of this paper is organized as follows: Section 2 reviews the related literature. Section 3 describes the problem studied and the main idea of our heuristic algorithm. Section 4 describes the procedure of the algorithm. Section 5 conducts extensive computational experiments and compares their results with those of Yalcin et al. (2019a). Section 6 concludes this paper by a discussion on major features of the algorithm and future research.

2. Related literature

The seminal work of PBS system was introduced by Gue and Kim (2007), they developed an optimal analytical result for the single-item retrieval problem in one escort PBS systems. The authors also considered the very specific scenario for multi escorts situation that all the escorts are horizontally arranged next to each other and adjacent to the I/O location in the bottom left corner of the system. They developed a dynamic program to solve this special configuration and provided optimal solutions for six cases of system size 5-by-9 with up to 6 escorts. They came up with a heuristic approach for larger size of this scenario. Kota, Taylor, and Gue (2010) offered an integer programming formulation to find the optimal retrieval time for general cases of multiple escorts, but not practicable because the time for solving the integer programming is unreasonable for even a very small PBS system. Five years later, Kota, Taylor, and Gue (2015) develop a closed-form expression for the retrieval time in PBS system with two escorts randomly distributed within the grid. For the situations when the number of escorts is more than two, they proposed a heuristic that can produce a near optimal solution, which provided the best result until Yalcin et al. (2019a) proposed an exact algorithm $MinMov$ and its heuristic variant, a very sophisticated design based on A* algorithm. The heuristic of Yalcin et al. (2019a) improved the average accuracy from over 10% to less than 3.5%, and in terms of CPU time, even the exact algorithm is 70 times fast than the heuristic algorithm of the Kota, Taylor, and Gue (2015) for an instance with grid size of 10×10. Yalcin et al. (2019b) created a framework for the evaluation of a PBS system based on a multi-agent routing algorithm, and a simulation-based case study of a grid-based baggage storage system at a major German airport demonstrated that such a system can achieve a storage density of up to 100% while being competitive in terms of retrieval time performance.

Unlike the normal SBS/RS system whose scheduling problem has been well solved, recent studies of the system are focused more on the analysis of its overall performance (Lerher, 2018; Jerman et al., 2021; Ekren et al., 2018), what we need urgently for the study of the PBS system is to solve the scheduling problem of its shuttle or AGV. Mirzaei et al. (2017) proposed an approach for simultaneous retrieval of multiple items in a 1-escort PBS system. They designed an optimal way for the retrieval of two items, and a heuristic method for three or more items. Furmans et al. (2011) investigated the PBS system with one vehicle and one escort, and focused on the issues of system design such as aspect ratio and I/O point location, etc. Alfieri et al. (2012) investigated a type of PBS system with limited number of AGV. They proposed a heuristic algorithm to optimize the movement of shelves and to dispatch the AGVs. These literatures studied the PBS systems in some simple special cases, which only account for a small part of actual application scenarios and cannot meet the requirements of a general PBS system in terms of efficiency and service capacity.

Gue et al. (2014) proposed a decentralized PBS system by introducing a negotiation scheme among grid cells. In the system, each row has at least one escort to ensure proper operation of the system and all the loads flow from one side to the other side. Yu et al. (2019) considered a situation of where block move is allowed, by employing integer programming, they obtained the optimal retrieval time of a single item in PBS system. Bukchin and Raviv (2020) proposed an exact dynamic programming algorithm, for the problem with two reconciling objectives of move time and move cost, this algorithm is only applicable to the problem with grid size smaller than 9×9.

Zaerpour et al. (2015; 2017a; 2017b) proposed a scenario of 3-D PBS system (Live-Cube), and assumed there are sufficient escorts available that a virtual aisle can be created in a situation, however, their works were more about strategic solutions than operational decisions, because they did not specify on how to optimally create a virtual aisle. The approaches of creating virtual aisles can ensure the speed of item retrieval, but it is also accompanied by two major limitations, on the one hand, it will cause unnecessary movement and therefore high energy consumption; on the other hand, there will be a higher demand for the number of escorts, which reduces the storage density. These two drawbacks are also existed in the decentralized control PBS systems, such as proposed by Gue et al. (2014). Although, the above drawbacks of virtual aisle can be alleviated partially by the integer programming model designed by Yu et al. (2019), which, by allowing block move, can find an optimal solution for minimizing item retrieval time at the cost of no guarantee of minimization of item moves even as a secondary goal.

Beam search is a heuristic search method that explores a graph of possible partial solutions by selecting a

limited number (beam size or beam width) of most promising partial solutions at each iteration. This method has been applied to cutting problems (Parreño et al, 2020), assembly line balancing (Li et al, 2021), container loading problem (Araya et al, 2020; Araya & Riff, 2014), scheduling problems (Birgin et al ,2020) and quay crane scheduling (Kress et al, 2019). Although beam search has been used to solve these combinatorial optimization problems with good performance, we have not found an application of beam search in scheduling PBS systems.

Therefore, many research opportunities are still existed to make better use of the attributes of the basic building block, i.e., M-SRPME, of the PBS system. The most critical issue of all is to find a better algorithm for M-SRPME, which, so far, the best result is the heuristic $MinMov$ algorithm designed by Yalcin et al. (2019a). We will use their research as a benchmark to further advance the solution accuracy and to reduce the CPU time consumption of the algorithm in the rest part of this paper.

3. Problem description and outline of the solution algorithm

3.1 Problem description and assumptions

In order to catch the main features of M-SRPME and evaluate the key performance of our algorithm for it, we ignore its variants that may exist in commercial applications, such as the positions and number of multiple I/O locations, which can be realized through some coordinate transformations and the adaptation of the algorithm. The assumptions that are made to specify the problem in this paper are summarized below:

(1) The system is unit loads, i.e., each location in the grid can only keep one item.

(2) The cost of moving an item to its adjacent location of escort is one.

(3) The single I/O location is at the lower left corner of the system.

(4) The system has m rows and n columns, both counted from the I/O location $(m = 1, n = 1)$.

(5) The objective of the problem is to minimize the total number of moves of the requested item (R) and all other items for the retrieval of item R.

(6) Backward move is not allowed. We ignore all backward-moves of item R in our heuristic and assume that item R can only be moved down or to left.

The purpose of the first five assumptions, which are the same as those made in Yalcin et al. (2019a), is to simplify the problem and make our algorithm comparable with theirs, while keeping the generality of the problem. The sixth assumption is a tactic of our heuristic to make a good tradeoff between the solution

accuracy and computational efficiency, which will be revi□ited in detail in □ection 3.2

To move a reque□ted item **(R Move)** to one of it□adjacent location□, the adjacent location mu□t be an empty location (e□cort). If the adjacent location i□ occupied by an item (thi□ item i□ called an **occupied item** hereafter), we mu□t fir□tly make a concatenated □equence of occupied item move□from an available e□cort to that adjacent location (**Escort move**), we refer to □uch concatenated path of occupied item-e□cort move□a□a **clearance path** (□ee Figure 1 (b, d, f)), and the co□t of forming a clearance path i□the **clearance cost**. After each move of item R, the prior location of item R become□an e□cort which referred a□the **base escort** (Yalcin et al., 2019). Obviou□ly, after the initial move, the large□t clearance co□t of the ba□e e□cort i□4. In 1-e□cort □ituation, a maneuver of 3-move□(□ee Figure 1 (d, e, f)) or 5-move□(□ee Figure 1 (f, g, h)) move□item R to the next adjacent location (Gue and Kim 2007), by the way of which an exact optimal □olution can attain.

In a PBS □y□tem with multiple e□cort□, utilizing an e□cort other than the ba□e e□cort may reduce the total number of item□move□and the ba□e e□cort in each □tep can be regarded a□a wor□t-ca□e alternative, by utilizing an e□cort with a □maller clearance co□t than that of the ba□e e□cort can '□ave' item move□and □uch an e□cort i□referred to a□a **saving escort** (Yalcin et al., 2019). Different moving direction□of item R and different utilization□of the e□cort□re□ult in different di□tribution of e□cort□and item R in the PBS □y□tem in each □tep, thi□di□tribution (layout) of e□cort□and item R i□referred to a□a **state** s of the □y□tem. Figure 1 □how□8 different □tate□and the location exchange□between e□cort□and item□.

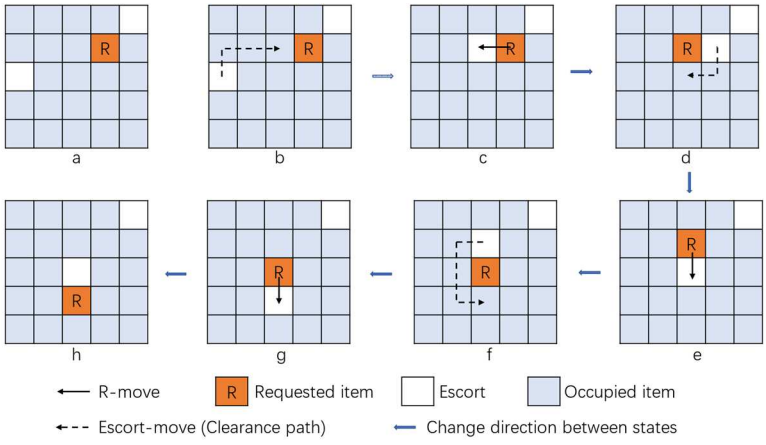

Figure 1. State□in a PBS and it□item movement□

3.2 Main idea of the algorithm

Although M-SRPME i□an extremely hard problem, it□□mall-□ize in□tance□have become tractable in recent year□. Therefore, we can u□e the propertie□of the □mall-□ize in□tance□to de□ign the neighborhood□of the

problem for our search algorithm. In order to do so, the optimal solutions of tens of thousands of randomly generated small-size instances of the problem obtained offline are exploited to fit an appraisal function for the neighborhoods of the system. Based on this, we propose a hybrid algorithm that combines state appraisal, neighborhood search and beam search. State appraisal provides the fitness value of the state derived by each possible next move, which is like the estimate of the total cost for guiding its search in the A* algorithm but the fitness value is much easier to compute and is accurate enough to well guide the neighborhood search in our algorithm. The neighborhood search finds a local optimum for one step of move (the next move) and calculates the cost of the current state and that of next state by the number of moves from the initial state. To avoid the state explosion, we adopt the beam search approach, only keeps some most promising partial solutions in each step of our algorithm. This can limit the number of states to be examined in each step and helps the algorithm to make a good tradeoff between solution accuracy/optimality and computation time.

Moreover, we observe that the cost of a solution with backward-moves (up or right moves) of the requested item R in a PBS system is usually much higher than forward-moves, thus the probability of a backward-move of R in the optimal solution of M-SRPME is extremely small. In fact, our experiments in section 5 show that the impact of considering backward-moves on the optimality of the solution obtained by our algorithm is only account for at most 0.073% on average in terms of cost. After making a tradeoff between the searching efforts saved by ignoring backward-moves and the tiny benefits gained by allowing backward-moves, we ignore all backward-moves of item R and assume that item R can only be moved down or to left for the simplification of our algorithm, that is no-backward-move assumption.

3.3 Definition of the neighborhood

The neighborhood of a PBS system defined in this paper has two properties: (1) Small enough for an exact search method to find the optimal (best) neighbor quickly. For instance, the method $MinMov$ proposed by Yalcin et al. (2019a) can be employed in the neighborhood search, whereby we can extract useful information to fit an appraisal function for the states of the system. (2) Big enough to include all escorts in one step move of item R in an optimal solution of the system. Although the search approach we propose in this paper is a heuristic one, we still do not want to miss opportunities of finding the optimal solution when it is possible in case of small instances for the problem. Since the largest clearance cost of the base escort is 4 after the initial move, we define the neighborhood as the area composed of all the locations with clearance cost less than or

equal to 3, plus the location of the base escort (see Figure 2), where the number in each square is the clearance cost from its location moved to item R's next move location, i.e., the square with 0 clearance cost. Since no backward-move is allowed in this paper, we only consider two forward moves in each step, i.e., left move and down move. Under the no-backward-move assumption and given the next move direction, the **backward zone** in the neighborhood refers to the area in which the clearance cost of each escort cannot be reduced regardless of the moving direction after the next move. The locations in the neighborhood other than those in the backward zone constitute the **forward zone**. (See Figure 2)

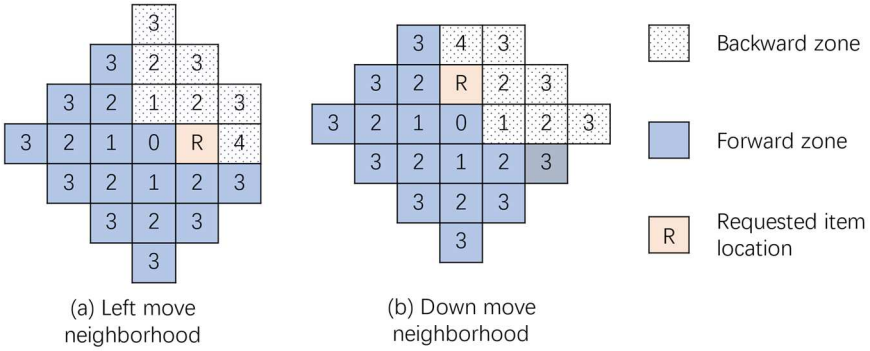

Figure 2. Neighborhood of the requested item

Theorem 1: After the initial move, the neighborhood defined by Figure 2 includes all the possible escorts in an optimal solution of M-SRPME at each step of item R move.

Proof: The escort optimally selected must have a smaller clearance cost than that of the base escort, and the maximum possible clearance cost of the base escort is 4, so the maximum possible clearance cost of a saving escort is 3. The neighborhood defined by (a) or (b) of Figure 2 includes all the escorts with clearance cost smaller than or equal to 3, and the base escort, hence includes all the possible escorts in an optimal solution of M-SRPME at each step of R move.

3.4 Definition of the notation and list of abbreviations

The symbols that will be used later in this paper are listed in Table 1 for reference, and some of them will be further explained where the corresponding concepts are introduced.

Table 1. List of notations

Symbol	Definition
$G = (L, E)$	Grid graph
$l = (i, j)$	A location in the grid with coordinate (i, j)
L	Set of locations in the grid

Symbol	Definition
$neighbours(l)$	Set of location□adjacent to location l
$forwardNeighbours(l)$	Set of location□adjacent to location l on it□left or down □ide
$backwardNeighbours(l)$	Set of location□adjacent to location l on it□right or up □ide
$e = (l, l'),$ $l' \in neighbours(l))$	Edge
l_0	I/O location
l_1	The □tart location of the reque□ted item
$d(l_1, l_2)$	The di□tance or number of move□from location l_1 to location l_2
$d^{l'*},\ l' \in neighbours(l))$	The minimum move co□t of clearance path□from all e□cort□to a next move location
$n^{l'},\ l' \in neighbours(l))$	The □ize of the clearance path□ □et of a next move location
L_E	Set of e□cort location□
$l_e \in L_E$	Location of an e□cort
n_e	Number of e□cort□
$\pi_E = \langle e_1, \dots, e_n \rangle$	Clearance path
$length(\pi_E)$	Length, or move co□t of a clearance path
$head(\pi_E)$	The head, or fir□t edge of a clearance path
$ending(e)$	Ending location of edge e
S_π^l	Set of clearance path□of location l
m, n	Grid □ize in $m \times n,\ m \le n$
$l_s \leftarrow location(s)$	The reque□ted item location at □tate s
$s = \{L_E, l_s\}$	State □
M_s	The motion plan of a partial □olution leading to □tate s from initial □tate
R_l	A rivet, which i□a □et of □tate□(a□□ociate with their partial □olution□(motion plan□)) with the □ame current reque□ted item location l
F_R	A frontier i□a □et of rivet□with all their □tate□' location□ l_s having the □ame Manhattan di□tance□to the I/O location
r	Maximum □ize of F_R, i.e., the maximum number of rivet□that can be kept in each frontier
p	Maximum □ize of R_l, i.e., the maximum number of □tate□(a□□ociated with their partial □olution□) that can be retained in each rivet
$l_R \leftarrow location(R_l)$	Location of rivet, i.e., current reque□ted item location of any partial □olution in rivet R_l
$L(F_R)$	The □et of location□of the rivet□on frontier F_R
$l_{HNF} \leftarrow NextLocation(F_R)$	Location of head rivet in next frontier which i□realized by a forward move of F_R, i.e., by moving the item at the top left location in □et $L(F_R)$ to left. If □uch move can't be made, the item will be moved down
L_N	Set of location□of the neighborhood of the reque□ted item
$l_{\tau,\lambda} \in L_N$	A location in the neighborhood, where τ i□the clearance co□t and λ i□the e□cort location index
$L_B \subset L_N$	Set of location□of the backward zone in the neighborhood
$L_{NL} \subset L_N,\ L_{ND} \subset L_N$	Set of location□of the left, down move neighborhood
$\vartheta,\ \vartheta^{left}, \vartheta^{down}$	Neighborhood □ize regulator; Left move, and down move neighborhood □ize regulator
$w_{l_e}^{left},\ w_{l_e}^{down}$	E□cort weight in the left move and down move, re□pectively, of the apprai□al neighborhood
d_N	E□cort□den□ity of a neighborhood
$F(d_N)$	The fitted function u□ed to calculate the overall e□cort weight of a neighborhood
$a(s)$	The value of □tate s a□□e□□ed by a neighborhood appra□□al procedure

Symbol	Definition
$g(s)$	Minimum move cost leading to state s from the initial state
$f(s) = a(s) - g(s)$	Fitness of state s
$f(R)$	The best fitness of all states in rivet R, which is the minimum of $f(s)$ over all states in the rivet
$C_s = \{s, M_s, f(s), g(s)\}$	A label associated with state s, its motion plan M_s and the values of $f(s)$ and $g(s)$

To facilitate reference, the following Table 2 provides a list of abbreviations for the key concepts/words used in this paper.

Table 2. List of abbreviations

Abbreviation	Detailed explanation
AGV	Automated Guided Vehicles.
I/O location	Input and output location.
$MinMov$	The exact algorithm developed by Yalcin et al. (2019b) for solving M-SRPME.
M-SRPME	The problem that minimizes the number of item-moves of SRPME.
PBS	Puzzle-based storage.
SBS/RS	Shuttle-based storage and retrieval systems.
SRPME	Single-item Retrieval Problem for PBS systems with Multiple Escorts.
T-SRPME	The problem that minimizes the retrieval time or the item move time of SRPME.

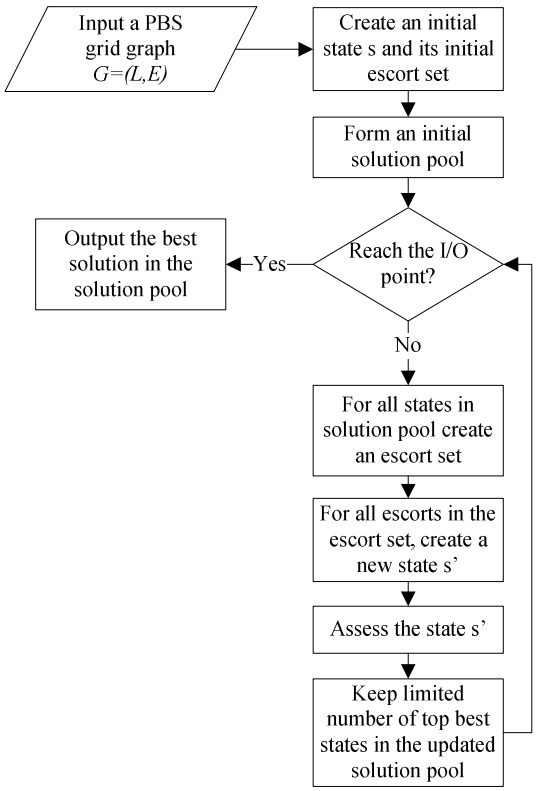

Figure 3. flow chart of the procedure of the heuristic

4. The algorithm procedure

In this section, we first introduce the Beam Search Procedure, which is also the main framework of our proposed heuristic algorithm. Then introduce the neighborhood search, which includes three subroutines in the algorithm, and finally analyze the computational complexity of the heuristic.

4.1 Beam Search Procedure

We incorporate a neighborhood search into beam search via a state appraisal function fitting and a beam size control tactic, which seeks a better balance between the solution accuracy and the computational efficiency. Beam Search acts as the main frame of our algorithm, it consists of a main beam search process and a frontier iterative process, both of which iteratively calculate/update the solution pool in the form of a frontier and several rivets (See Figure 3).

4.1.1 Main beam search process

The solution pool, with the function of beam size controlling, is defined as a frontier which consists of one or more rivets, where each rivet is a subset of the frontier (See Figure 4). Since backward-move is not allowed in the heuristic, the frontier is moving down left toward the I/O point in each iteration. The detailed procedure of this process is given in the pseudo code of Algorithm 1, $BeamSearchPBS$, which is also an expanded description of Figure 3.

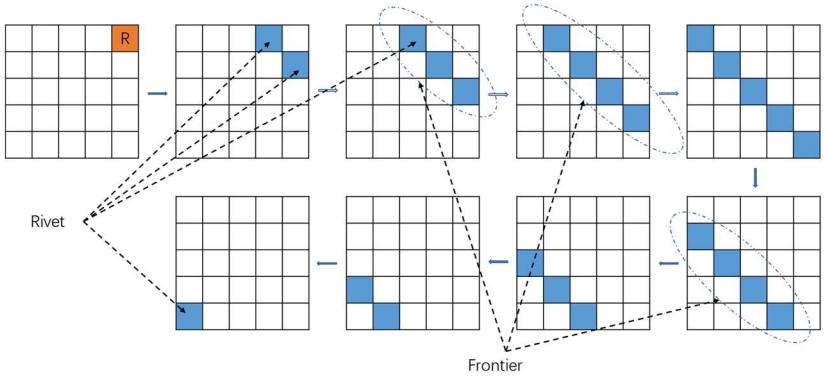

Figure 4. Illustration of the rivets and frontier while the heuristic algorithm iterates

We define and initialize a global state variable, label C_s to hold and keep the states and solution data updated during the execution of $BeamSearchPBS$ and its subroutines Algorithm 2 to 4 (line 1). In Algorithm 1, Line 2 to line 4 check if the goal I/O location is reached, then return the initial motion plan, which means an empty set with zero move cost. Line 5 considers all available escorts to create two sets of

clearance paths for the next R locations. The Dijkstra Algorithm (Dijkstra, 1959) or the Maximum Nearest Escort Search algorithm proposed by Yalcin et al. (2019a) can be used here to find the shortest path (minimal moving cost) between two locations.Yalcin et al. (2019a) has proved that at most six best escorts with move cost less than or equal to the lowest clearance cost plus 3 are needed to consider in a search towards an optimal solution at the initial escort move step, so we adopt the same rule at the initial move step (lines 6-8). Line 10 to line 19 build the initial solution pool by exploring all the forward move directions for all the clearance paths in set S_π and generating new states, then assess each state in the pool by the subroutine $StateAppraisal$ (Algorithm 4) (line 13), and finally control/limit the size of solution pool rivet R_l(line 16) and frontier F_R (line 19). Line 21 to line 24 is the frontier loop which will be executed if the goal location is not reached, where $FrontierIteration$ (Algorithm 2) is a subroutine that moves a frontier forward by one step. When the algorithm iterates to the second last frontier, l_{HNF} will be at the I/O point (line 21), and the information of all the states till the last frontier will be stored in the set F_R. When the frontier loop ends, there are 2 rivets in the frontier and the best one is chosen (lines 25 and 26).

Algorithm 1. $BeamSearchPBS(G, L_E, l_1, r, p)$

Input:

Grid graph $G = (L, E)$, the initial set of escort locations L_E, the initial location l_1 of the requested item, the maximum number of rivets r and the maximum number of states p.

Output:

M_s, a motion plan that moves item R from $l_1$to the I/O point

1: create initial state s with $l \leftarrow l_1$ and L_E, $M_s \leftarrow \emptyset$, $g(s) \leftarrow 0$,
$f(s) \leftarrow \infty, C_s \leftarrow \{s, M_s, f(s), g(s)\}$ ▷ C_s is a global state variable

2: if $l = l_0$ then ▷ Check if the goal location is reached

3: return M_s

4: end if

5: $S_\pi^{l'} \leftarrow$ set of all the clearance paths between $l \in l_E$ and $l' \in forwardNeighbours(l)$ ▷ Initial move

6: $d^{l'*} \leftarrow \min_{\pi_E' \in S_\pi^{l'}} length(\pi_E')$

7: $n^{l'} = \min(6, the\ number\ of \pi_E in\ S_\pi^{l'}$
$and\ length(\pi_E)$ with $length(\pi_E) \leq d^{l'*} + 3)$

8: Keep in $S_\pi^{l'}$ only the $n^{l'}$ shortest clearance paths

9: $F_R \leftarrow \emptyset$, ▷ Initial frontier generation

10: for all $l' \in forwardNeighbours(l)$ do

11: $R_l \leftarrow \emptyset$

12: for all $\pi_E \in S_\pi^{l'}$ do

13: $C_{s'} \leftarrow StateAppraisal(C_s, l', \pi_E)$ ▷ Assess each new state

14: $R_l \leftarrow R_l \cup C_{s'}$

15: end for

16: keep best p states in R_l ▷ Choose best states according to the value of $f(s)$
17: $F_R \leftarrow F_R \cup R_l$
18: end for
19: keep best r rivets in F_R ▷ Choose best rivets according to the value of $f(R)$
20: $l_{HNF} \leftarrow NextLocation(F_R)$
21: while $l_{HNF} \neq l_0$ do ▷ While the goal location is not reached
22: $F_R \leftarrow FrontierIteration(F_R, r, p)$ ▷ Frontier loop
23: $l_{HNF} \leftarrow NextLocation(F_R)$
24: end while
25: $R_l^* \leftarrow argmin_{R_l \in F_R} f(R)$
26: return $M_s \leftarrow argmin_{s' \in R_l^*} g(s')$

4.1.2 Frontier iteration

With the assumption of no-backward-move, the search effort of our heuristic is greatly reduced to at most $2n-3$ iterations (see Figure 4). In each frontier iteration, a new frontier is created from the last frontier, all the states in the solution pool of the last frontier are updated by Algorithm 3 and assessed by Algorithm 4 successively, and a solution pool for the new frontier is created by retaining some best states. Each frontier iteration provides a container (pool) for possible solutions, and the states in the same frontier are comparable in terms of Manhattan distance from their requested item location to the I/O location. During each frontier iteration, the evolution of states and their performances ($g(s)$ and $f(s)$), as well as their corresponding partial solutions (motion plan M_s), are stored implicitly in label C_s of Algorithm 2.

Algorithm 2. $FrontierIteration(F_R, r, p)$

Input:

Previous frontier F_R with set of rivets, where the number of rivets is limited by r, and the number of states in a rivet is limited by p.

Output:

New frontier F_R' with set of rivets

1: $F_R' \leftarrow \emptyset$ ▷ Initialize new frontier
2: for all $R_l \in F_R$ do
3: for all $l_R' \in forwardNeighbours(location(R_l))$ do
4: $R_{l'} \leftarrow \emptyset$ with location of l_R'
5: $F_R' \leftarrow F_R' \cup R_{l'}$
6: end for
7: end for
8: for all $R_{l'} \in F_R'$ do ▷ Rivet loop
9: $l' \leftarrow location(R_{l'})$
10: for all $l \in backwardNeighbours(l')$ && $l \in L(F_R)$ do
11: for all $s \in R_l$ do
12: $S_\pi^{l'} \leftarrow EscortSearch(s, l, l')$
13: for all $\pi_E' \in S_\pi^{l'}$ do ▷ Candidate escorts loop

14:	$C_{s'} \leftarrow StateAppraisal(C_s, l', \pi'_E)$	▷ Assess each new state
15:	$R_{l'} \leftarrow R_{l'} \cup C_{s'}$	
16:	end for	
17:	end for	
18:	end for	
19:	keep best p states in $R_{l'}$	▷ Choose the best states according to the value of $f(s)$
20:	end for	
21:	keep best r rivets in F'_R	▷ Choose the best rivets according to the value of $f(R)$

Algorithm 2. $FrontierIteration$ conducts frontier iteration and completes the one-step transformation from previous frontier to the new one. Line 1 to line 7 initialize a new frontier, generates an empty frontier with empty set of states, and initialize the size and the locations of the frontier. All states in each rivet in the previous frontier are extended to new states by using their available escorts (lines 8-21). Algorithm 2 Assesses and saves each new state in the new rivets and new frontier, and finally controls the size of the solution pools of rivets and the new frontier. The subroutine $EscortSearch$(Algorithm 3) finds a local optimum set of escorts and its clearance path for a given state (line12), and the subroutine $StateAppraisal$(Algorithm 4) assess each new state (line 14).

4.2 Neighborhood search

Neighborhood search deals with the changes between states, it includes two subroutines: (1) escort search, to find a suitable escort to swap with the next move location to form a new state. (2) state appraisal, to assess the fitness value of each new state.

4.2.1 Escort search

Given a next move location l' of a state, $EscortSearch$ (Algorithm 3) searches for a set of escorts and then builds a clearance path for each escort from its current location to the next move location of the state. In order to facilitate the reference and save the computation time of the clearance cost, we code each escort location in the neighborhood (see Figure 5) by two numbers separated by a comma, where the number before the comma is the clearance cost τ, and the number after the comma is the escort location index λ, and the locations in the neighborhood are referred as $l_{\tau,\lambda} \in L_N$, where L_N is the set of locations in the neighborhood of the requested item.

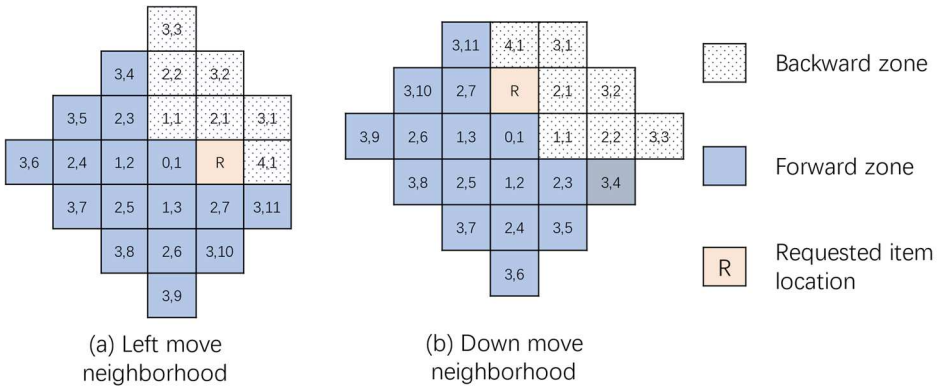

Figure 5. The zoning and coding of each escort location in the neighborhood

Observation 1: Under the assumption of no-backward-move, for any escort in the backward zone of the neighborhood of a state, its clearance cost will increase as the state evolves into any future state.

We can observe the result of this observation by moving the next move location downward or leftward. In both scenarios, the conclusion of observation 1 is true, and the farther the escort is, the higher its clearance cost becomes.

Assume that an escort E in the backward zone of the neighborhood of the current state has the smallest clearance cost among all escorts in the neighborhood, according to Observation 1, the opportunity cost of not using E in the current state will increase in the future, therefore E should be used first. (Case 1)

In Figure 5, the location with 0 distance from the current requested item's next move location is l'. The pseudo code of Algorithm 3 involves two scenarios, which are left move and down move, respectively. For the sake of brevity, we only show the two scenarios in a general way in Algorithm3.

Algorithm 3. $EscortSearch(s, l, l')$

Input:

State s and its location l, and the next move location l'

Output:

Set of clearance path S_π

1: $S_\pi \leftarrow \emptyset$, and define L_N and L_B according to location l, l' and Figure 5. ▷ initialization
2: if $l' \in L_E$ then
3: $\pi_E \leftarrow \emptyset$
4: return
5: else ▷ Case 1
6: $l_e^* = arg \min_{l_e \in L_E \cap L_B} d(l_e, l')$
7: if $d(l_e^*, l') \leq \min_{l_e \in L_E \cap L_N \setminus L_B} d(l_e, l')$ then
8: $\pi_E \leftarrow$the clearance path form l_e^* to l'
9: $S_\pi^{l'} \leftarrow S_\pi^{l'} \cup \pi_E$
10: return
11: end if
12: end if

13: if $l_{2,1} \in L_E$ or $l_{2,2} \in L_E$ then ▷ Case 2
14: $S_\pi^{l'} \leftarrow$ set of clearance paths for all $l_e \in l_N$ and $d(l_e, l') \leq 2$
15: $n = \min(3, the\ number\ of \pi_E \in S_\pi^{l'})$
16: else if $l_{3,1} \in L_E$ or $l_{3,2} \in L_E$ or $l_{3,3} \in L_E$ then ▷ Case 3
17: $S_\pi^{l'} \leftarrow$ set of clearance paths for all $l_e \in l_N$ and $d(l_e, l') \leq 3$
18: $n = \min(4, the\ number\ of \pi_E \in S_\pi^{l'})$
19: else ▷ Case 4
20: $S_\pi^{l'} \leftarrow$ set of clearance paths for all $l_e \in l_N$
21: $n = \min(5, the\ number\ of \pi_E \in S_\pi^{l'})$
22: end if
23: keep in $S_\pi^{l'}$ only the n shortest clearance paths

In order to reduce unnecessary state generation as much as possible, we consider four cases to limit the number of escorts in Algorithm 3. Line 1 defines the set of locations of the neighborhood of the current state and its backward zone based on relative position of l and l' in Figure 5. If the next move location is an escort, R move can be done immediately, therefore the clearance path is an empty set, and return (lines 2-4). If an escort in the backward zone has the lowest clearance cost in the neighborhood, then use it and return according to Observation 1 (lines 5-12). Yalcin et al. (2019a) proved that in the situations of 3-move, 4-move and 5-move, corresponding to Case 2, Case 3 and Case 4 in $EscortSearch$ respectively, we only need to consider at most 3, 4, and 5 escorts, respectively, whose clearance costs are smaller than or equal to the clearance cost of the corresponding base escort (lines 13-23). The base escorts here are $l_{2,1}$, $l_{3,1}$, and $l_{4,1}$, respectively. In case 2, because $l_{2,2}$ is also in the backward zone and has the same clearance cost and opportunity cost as $l_{2,1}$ does, consequently, $l_{2,2}$ is equivalent to $l_{2,1}$ and can be used as the base escort in the no-backward-move situation (line 13). In case 3, the same equivalence relationship exists among $l_{3,1}$, $l_{3,2}$ and $l_{3,3}$ (line16).

4.2.2 State appraisal

Algorithm 4, $StateAppraisal$, generates and assesses a new state s', and returns its fitness value together with the new state in the form of a label $C_{s'}$. Firstly, this algorithm obtains the location of the selected escort from the set of clearance paths, and excludes it from the set of escort locations in the new state, and the next move location of the last state becomes the requested item location of the new state (line 1). Then, this algorithm defines a new neighborhood by its locations of left moves and down moves and generates a new state for subsequent calculations (lines 2-3), and update the set of motion plans (line 4) as well as the cost of the current motion plan (line 5). Line 6 realize the assessment operation for the two states of the forward neighborhood and choose the bigger one as the state assessment value, then the fitness and the label of the

new state are updated (lines 7-8).

Our heuristic must keep two frontiers in memory at each iteration, but does not need to assess the new states at the final iteration due to its invalid forward neighborhood. When item R approaches the edge of the grid, one or two exploring directions may become unavailable, then the corresponding appraisal value should be set as a negative infinity. The fitted function $F(\cdot)$ with the weights of the escorts defined in line 6 will be explained immediately below.

Algorithm 4. $StateAppraisal(C_s, l', \pi_E)$

Input:

the label C_s of state *s*, next move location l', and clearance path π_E

Output:

New solution collection set $C_{s'}$

1: $L'_E \leftarrow L_E \cup l_s \setminus \{ending(head(\pi_E))\}$, and $l_{s'} \leftarrow l'$ ▷ get the new escorts set and the location of new state

2: define L_{NL} and L_{ND} according to locations l_s, l' and figure 2

3: $s' \leftarrow \{L'_E, l_{s'}\}$

4: $M_{s'} \leftarrow concatenate(M_s, \pi_E, \langle l, l' \rangle)$

5: $g(s') \leftarrow g(s) + length(\pi_E) + 1$

6: $a(s') \leftarrow \max \begin{pmatrix} F\left(\sum_{l_e \in L'_E \cap L_{NL}} w_{l_e}^{left}\right)\vartheta^{left}, \\ F\left(\sum_{l_e \in L'_E \cap L_{ND}} w_{l_e}^{down}\right)\vartheta^{down} \end{pmatrix}$ ▷ appraisal the density of escorts

7: $f(s') \leftarrow a(s') - g(s')$

8: $C_{s'} \leftarrow \{s', M_{s'}, f(s'), g(s')\}$

1) Appraisal function $F(\cdot)$

The cost estimation for each move of item R is critical and very time consuming when searching for a good solution for M-SRPME. An A* algorithm-based shortest path method is employed to estimate the move cost from the current state to the goal state in the algorithms of Yalcin et al. (2019a), with the computational complexity of $O(n^2)$. We propose an assessment procedure to replace this time-consuming process. The assessment is based on an appraisal function which is fitted by utilizing the optimal solutions of a group of small-size instances of M-SRPME obtained offline.

Yalcin et al. (2019a) reported optimal solutions of tens of thousands of randomly generated small-size instances of M-SRPME, the size of these instances is ranged from n = 5 to n = 10. The neighborhoods we have defined in Figure 2 are 6×7 in size, which are trimmed grids area and smaller than size 7×7, we just use the optimal solutions of the 6×6 instances considered in Yalcin et al. (2019a) to approximately fit the appraisal function as a third-order polynomial:

$$F(d_N) = 22.394 d_N^3 - 67.005 d_N^2 + 66.818 d_N^1 - 1.9879 \quad (1)$$

where $d_N = n_e/(n^2 - 1)$, and n_e is the number of escorts.

d_N can be regarded as the density of escorts. Obviously, its feasible region is $(0,1]$. We assume that the larger the escorts density the better is the new state, that is, the greater d_N is, the higher the value of $F(d_N)$ is. Based on this assumption, function (1) should be greater than or equal to zero and strictly increase in the value of d_N. However, due to the random way of the instance's generation, this prerequisite of the appraisal function cannot be guaranteed. Therefore, we remove the constant term from function (1), so that the appraisal function can meet the requirements as much as possible, and avoid the situation of negative value of $F(d_N)$ when d_N approaches 0. It should also be noted, when item R reaches the border of the grid, the size of the neighborhood will be shrunken, we introduce a regulator ϑ to neutralize the impact of the change of the neighborhood size on the evaluation results by dividing d_N by ϑ. Thus, in our algorithm, we use function (2) as the appraisal function instead.

$$F(d_N) = 22.394\left(\frac{d_N}{\vartheta}\right)^3 - 67.005\left(\frac{d_N}{\vartheta}\right)^2 + 66.818\left(\frac{d_N}{\vartheta}\right)^1 \quad (2)$$

where ϑ is the sum of the weights of the escorts within the shrunken neighborhood, and the escorts weights will be explained in the following section.

2) The weights of escorts

When calculate the escort density, it is not appropriate to simply divide the number of escorts by the total number of locations of a neighborhood. When estimating move efficiency in a neighborhood, different escorts should be given different weights because their different clearance costs and location zones. We weigh more on an escort with a smaller clearance cost or in the forward zone. Taking the left-move neighborhood as an example, firstly, based on the clearance cost of each location (see Figure 2 and Figure 5), 0.5 is added to the weight of each location in the area on the upper side of R (including the row of R), the area on the right side of R (including the column of R), and the area on the upper right of R (including the row and column of R), respectively. In this way, 0.5 can be added to the weight of each location in the upper right area of R three times. Secondly, use the maximum possible weight 5.5 (=4+3×0.5, where 4 is the maximum possible clearance cost, i.e., the base escort) to subtract the weight of each location obtained so far to get its new weight, regardless of the weight of the location of R. Finally, the weights of all locations in the neighborhood are normalized to obtain the final weight of each location such that their total weight is equal to one (see Figure 6).

This method of determining the weight of each escort is somewhat arbitrary, but it has been proved effective by our numerical experiments, even though there may be a better method for setting escort weights such as by

machine learning. When a□e□ing the fitne□□of a □tate in □olving M-SRPME by our heuri□tic, we define the □um of total weight of all the □tate'□e□cort□in the neighborhood a□the e□cort den□ity d_N, and adju□t the value $F(d_N)$ by multiply it by the neighborhood □ize regulator ϑ when the neighborhood i□incomplete in ca□e it approache□the border of the underlying grid.

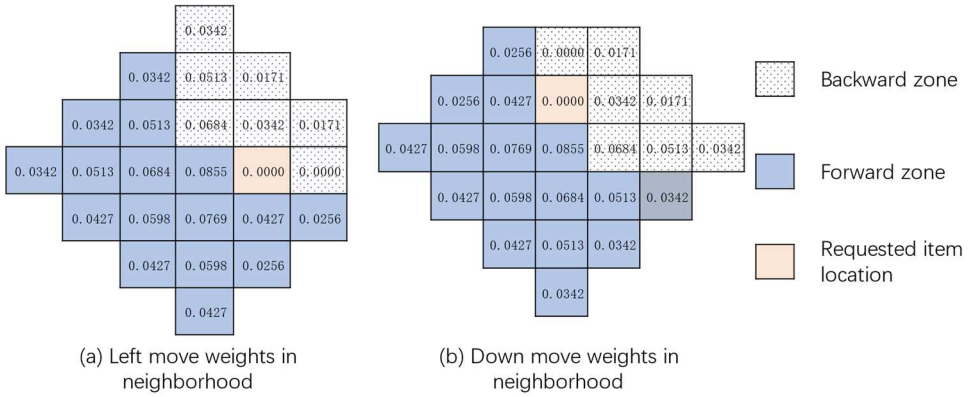

Figure 6. Weight□of e□cort location□in a neighborhood for □tate appra□□al

4.3 An illustrative example

The example in□tance i□of □ize 5×5 with the reque□ted item R located at the upper right corner of the grid graph, to be moved to the I/O point located at the bottom left corner of the grid graph (□ee the initial □tate in Figure 7). The procedure $BeamSearchPBS$, with beam □ize of $r = 2, p = 3$, yield□the motion plan with 18 move□, of which 10 move□are e□cort-move□and 8 move□are R-move□. The motion plan i□proved to be optimal by u□ing $MinMov$ of Yalcin et al. (2019a) to □olve thi□in□tance.

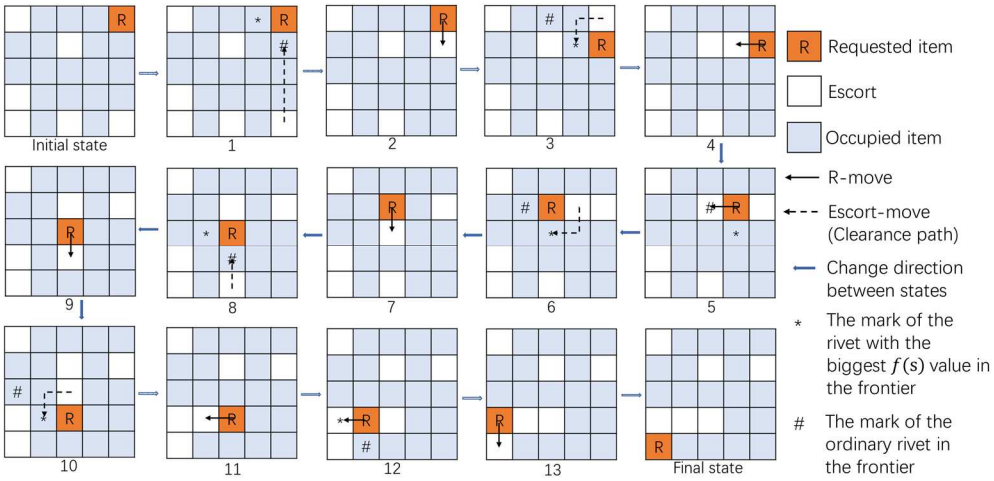

Figure 7. The motion plan obtained by running $beamSearchPBS$ with $r = 2, p = 3$

There are total 7 frontier□in thi□□olution, but not every time the reque□ted item i□moved to the location who□e rivet ha□the be□t fitne□□in it□frontier, □uch a□□tate□1, 5 and 8 in Figure 7, becau□e the off□pring □tate

expanded from the current best state is not necessarily the best, although it has a relatively large chance to be the best in the next iteration. If the values of r and p are both set to 1, that is, each iteration only retains the best state to expand, then $BeamSearchPBS$ becomes a typical greedy algorithm, the motion plan of this setting is 19 moves. In this situation, the requested item will always be moved to the location of the rivet with the best fitness value in the frontier. This example shows that the beam size of $BeamSearchPBS$ plays a key role in jumping out of local optimum, it also shows the poor effect of simply applying a greedy tactic in solving M-SRPME. For a more detailed explanation of this example, please see Appendices A1.

4.4 Computational complexity analysis

In order to capture the main aspects of the algorithm complexity, we only analyze the major loop of the algorithm, that is the while loop in algorithm 1 (line 21 to line 24), and its subroutines. Firstly, we set $r < p$ for the two parameters of our algorithm (see Section 5.1 and AppendicesA2 for why we set *r* and *p* in this way) and define two additional parameters as follows:

(1) k, the number of escorts, or clearance paths, considered in each candidate escort loop (line13 to 16 in Algorithm 2.)

(2) c, the constant computing time of subroutines Algorithm 3 and Algorithm 4. Because the operations of both algorithms are performed in a neighborhood, and the computational complexities of all their operations are linear with bounded parameters, so the computing time of the two subroutines can be considered a constant.

The outermost loop is the while loop in algorithm 1 (line 21 to 24), there are at most $m + n - 2$ ($m + n - 3$ when $n \geq m \geq 3$) frontier moves.

In frontier iteration (Algorithm 2), the major loop is the rivet loop (line 8 to 20), there are at most r iterations. At most 2 directions needed in the for loops (lines 10-18). There are at most p iterations in a rivet (lines 11-17), and at most k iterations in a set of clearance path (lines 13-16). By considering the computation time of sorting (lines 19, 21), escort searching (line 12), state appraisal (line 14), but ignoring the initialization operations (lines 1-7) with a lower-order computational complexity, we obtain the complexity expression of the algorithm 2 as:

$$2r(p(c + kc) + pk \ln(pk)) + 2r \ln(2r) \tag{3}$$

Where $2r$ is 2 move directions and r iterations of rivet. $p(c + kc)$ is p iterations in a rivet, c is the

constant time of escort search in line12, and kc is k times of escort constant appraisal time (lines 13-16). $pk\ln(pk)$ is the sorting time needed to find the best p states in pk states (line 19). $2r\ln(2r)$ is the sorting time needed to find the best r rivets in $2r$ rivets (line 21).

Combined with $m+n-2$ frontier moves of while loop, the outer most loop in Algorithm 1, the complexity expression of our heuristics is

$$(m+n-2)(2r(p(c+kc)+pk\ln(pk))+2r\ln(2r)) \tag{4}$$

The first term $m+n-2$ can be estimated as $2n$ given that $m \le n$, after doing some adjustment operations, function (4) is turned into:

$$4nr(p(c+k(c+\ln(pk))+\ln(2r)) = 4nr(p(c+k(c+\ln(p)+\ln(k))+\ln(2r)) \tag{5}$$

From Table 1 and Figure 4, we know $r \le m \le n$, and k can be considered a constant since it is smaller than or equal to 6, then $ln(2r)$, $ln(k)$, and c in (5) can be ignored, leading to the complexity of the heuristic:

$$O(4nrpk\ln(p))$$

By further ignoring the constant 4 and the constant parameter k, the complexity of our heuristic becomes:

$$O(nrp\ln(p)) \tag{6}$$

5. Computational experiments

5.1 The parameter design of our heuristic

The computational complexity expression (6) of our heuristic has three parameters, it would be easier for intuitive understanding if it was simplified to depend on only one parameter. In addition, it is also conducive to design reasonable parameters in real-world applications to control the computational time of the heuristic by setting appropriate values for r and p. To this end, we developed a mathematical model to appropriately set r and p as functions of n and calculated their corresponding computational complexity. Due to space limitation, we describe this model in Appendices A2, and get the five computational complexity settings of our heuristic, namely $O(n)$, $O(n^{1.5})$, $O(n^2)$, $O(n^{2.5})$ and $O(n^3)$.

5.2 Experimental problem design and computing environment

We designed two types of experimental instances. The first type contains the small-size instances generated in the same way as that of Yalcin et al. (2019a) with the grid sizes ranged from 2 to 10, mainly used to compare

the solution accuracy of our heuristic with the exact algorithm proposed by them. The second type is the large-size instances we generated to compare our heuristic with their heuristic.

According to whether the location of the requested item R is in the upper right corner of the grid or is randomly distributed in the grid, the small-size instances are divided into collection 1 and collection 2, respectively. A total of 41437 small-size instances in collection 1, 20455 small-size instances in collection 2 and 2300 large-size instances are generated. According to the grid size or/and the number of escorts of the instance, the small-size and large-size instances are further classified into several groups. See Appendices A3 for the detailed way to generate all instances in our numerical experiments.

Our heuristic algorithm was coded in MATLAB 2016b academic setting. Because we could not obtain the code of the exact and heuristic algorithms of Yalcin et al. (2019a), we recoded them in MATLAB to make our experimental results comparable with theirs. The experiments were conducted on a HP desktop with Windows 10 of home setting, 3.20 GHz Intel Core i5-6500U processor and 8 GB of RAM.

5.3 Experimental results

5.3.1 Results for small-size instances

We compared the computational results of our heuristic $BeamSearchPBS$ with five complexity settings with those of Yalcin's heuristic. The optimal solution of each instance is obtained by running $MinMov$ of Yalcin et al. (2019a), and the average relative percentage deviation of solution accuracy is calculated as:

$$ARPD = \frac{100}{G}\sum_{\forall g}\frac{\sum_{\forall i} H_{g,i}-\sum_{\forall i} O_{g,i}}{\sum_{\forall i} O_{g,i}} \quad (7)$$

where G means the total number of groups for a given grid size, $H_{g,i}$ is the solution value obtained by $BeamSearchPBS$ or Yalcin's heuristic for instance i in group g. $O_{g,i}$ means the optimal solution value obtained by $MinMov$ for instance i in group g.

1) The solution accuracy analysis

The results in Table 3 and 4 show that $BeamSearchPBS$ under all the computational complexity settings outperforms Yalcin's heuristic with computational complexity $O(n^4)$. The summarized results of these two tables are given in Table 7 in this section, which shows that our algorithm with computational complexity $O(n^3)$, $O(n^2)$, and $O(n)$ respectively can improve the overall average solution accuracy of all instances tested from Yalcin's 1.096% to 0.055%, 0.107%, and 0.570%, respectively, where n is the size of the PBS system. For PBS systems with high storage density, our algorithm shows stronger robustness by improving the

accuracy from Yalcin's 1.086% to 0.026%, 0.051%, and 0.249% by its complexity setting $O(n^3)$, $O(n^2)$, and $O(n)$, respectively.

From Table 3 and Table 4, we can also see that the solution accuracy of both algorithms deteriorates as the grid size increases. The main reason is that the estimation function $h(s)$ used in the two algorithms will become more and more inaccurate as the problem size increases, just as the accuracy of a long-term forecast is lower than that of a short-term one. However, the deterioration rate of $BeamSearchPBS$ slows down as its computational complexity increases. Please also see (A) and (B) in figure 8 for the relationships between the instance size and the solution accuracy for both algorithms.

Table 3. Overall solution accuracy comparison ($ARPD$) of the two heuristics for small-size instances - collection 1

n	G	I	$YalH$	$O(n)$	$O(n^{1.5})$	$O(n^2)$	$O(n^{2.5})$	$O(n^3)$
2	3	7	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
3	8	255	0.376%	0.000%	0.000%	0.000%	0.000%	0.000%
4	15	4011	1.042%	0.056%	0.026%	0.009%	0.000%	0.000%
5	24	8201	1.205%	0.288%	0.076%	0.021%	0.008%	0.001%
6	35	19261	1.403%	0.667%	0.203%	0.071%	0.030%	0.010%
7	48	5751	1.508%	0.936%	0.179%	0.104%	0.046%	0.014%
8	15	1875	1.587%	1.270%	0.284%	0.155%	0.072%	0.022%
9	16	1126	1.553%	1.429%	0.377%	0.195%	0.093%	0.029%
10	19	950	1.769%	1.753%	0.532%	0.208%	0.104%	0.053%
*sum/#Avg.	183*	41437*	1.160%#	0.711%#	0.186%#	0.085%#	0.039%#	0.014%#

n: grid size $(n \times n)$, G: number of groups, I: number of instances, $YalH$: Yalcin's heuristic, $O(n), O(n^{1.5}), \cdots, O(n^3)$: $BeamSearchPBS$ with five computational complexity settings respectively, #: simple average of the data above in the corresponding column of the table

Table 4. Overall solution accuracy comparison ($ARPD$) of the two heuristics for small-size instances – collection 2

n	G	I	$YalH$	$O(n)$	$O(n^{1.5})$	$O(n^2)$	$O(n^{2.5})$	$O(n^3)$
5	24	2325	0.718%	0.052%	0.016%	0.012%	0.012%	0.012%
6	35	3436	0.893%	0.214%	0.106%	0.056%	0.048%	0.045%
7	48	4749	0.997%	0.423%	0.110%	0.088%	0.068%	0.061%
8	63	6264	1.202%	0.533%	0.213%	0.159%	0.136%	0.121%
9	17	1681	1.061%	0.562%	0.248%	0.193%	0.164%	0.139%
10	20	2000	1.317%	0.790%	0.364%	0.272%	0.205%	0.196%
*sum/#Avg.	207*	20455*	1.031%#	0.429%#	0.176%#	0.130%#	0.105%#	0.096%#

n: grid size $(n \times n)$, G: number of groups, I: number of instances, $YalH$: Yalcin's heuristic, $O(n), O(n^{1.5}), \cdots, O(n^3)$: $BeamSearchPBS$ with five computational complexity settings respectively, #: simple average of the data above in the corresponding column of the table

2) Backward-move impact analysis

Algorithm $BeamSearchPBS$ is based on the hypothesis of no-backward-move of requested item, it is necessary to analyze the impact of this assumption.

We checked every motion plan of all the 41437 instances in the first collection of small-size instances solved

by $MinMov$, find that there is no instance with backward move in its optimal motion plan, this may be because the requested item is located at the upper right corner of the grid. Table A.4 in Appendices A4 shows the impact of no-backward-move on the solution accuracy of small-size instances in the second collection, whose requested item can be located anywhere initially and may result in an extreme situation where one or two steps of backward move could generate a little cost savings. There are only 181 out of 20455 instances with 1 or 2 backward moves, most of the 181 instances have a single backward move, only 8 out of the 181 instances have 2 backward moves. and the overall impact of no-backward-move on solution optimality is only about 0.073%. (See Table A.4 in Appendices A4)

3) The algorithms' CPU time analysis

The average CPU time consumption by the two heuristics is calculated as:

$$ACT = \frac{1}{G}\sum_{\forall g}\left(\frac{1}{I_g}\sum_{\forall i} T_{g,i}\right) \quad (8)$$

where G means the total number of groups for a given grid size, I_g means the number of instances in group g, $T_{g,i}$ is the CPU time obtained by an algorithm for instance i in group g.

Table 5 and Table 6 show that in all complexity settings, $BeamSearchPBS$ run faster than Yalcin's heuristic. Please also see figure (C) and (D) in figure 8 for the relationships between the instance size and the average CPU time.

Table 5. Average CPU time in seconds (ACT) for solving small-size instances - collection 1

n	G	I	$YalH$	$O(n)$	$O(n^{1.5})$	$O(n^2)$	$O(n^{2.5})$	$O(n^3)$
2	3	7	0.028	0.043	0.007	0.012	0.023	0.021
3	8	255	0.109	0.065	0.085	0.085	0.089	0.092
4	15	4011	0.534	0.123	0.177	0.232	0.311	0.399
5	24	8201	1.687	0.190	0.301	0.462	0.692	1.072
6	35	19261	4.179	0.256	0.434	0.750	1.221	2.135
7	48	5751	9.018	0.318	0.738	1.096	1.934	3.718
8	15	1875	17.462	0.385	0.937	1.613	2.948	5.883
9	16	1126	28.638	0.445	1.112	2.086	4.022	8.561
10	19	950	46.912	0.508	1.296	2.588	5.458	12.140

n: grid size $(n \times n)$, G: number of groups, I: number of instances, $YalH$: Yalcin's heuristic, $O(n), O(n^{1.5}), \cdots, O(n^3)$: $BeamSearchPBS$ with five computational complexity settings, respectively

Table 6. Average CPU time in seconds (ACT) for solving small-size instances - collection 2

n	G	I	$YalH$	$O(n)$	$O(n^{1.5})$	$O(n^2)$	$O(n^{2.5})$	$O(n^3)$
5	24	2325	0.677	0.061	0.080	0.105	0.142	0.189
6	35	3436	1.804	0.086	0.119	0.169	0.249	0.375
7	48	4749	2.800	0.116	0.199	0.258	0.411	0.688
8	63	6264	4.911	0.141	0.255	0.364	0.601	1.050
9	17	1681	7.843	0.166	0.314	0.467	0.810	1.516

10	20	2000	12.709	0.197	0.394	0.610	1.130	2.283

n: grid size $(n \times n)$, G: number of groups, I: number of instances, $YalH$: Yalcin's heuristic, $O(n), O(n^{1.5}), \cdots, O(n^3)$: $BeamSearchPBS$ with five computational complexity setting, respectively

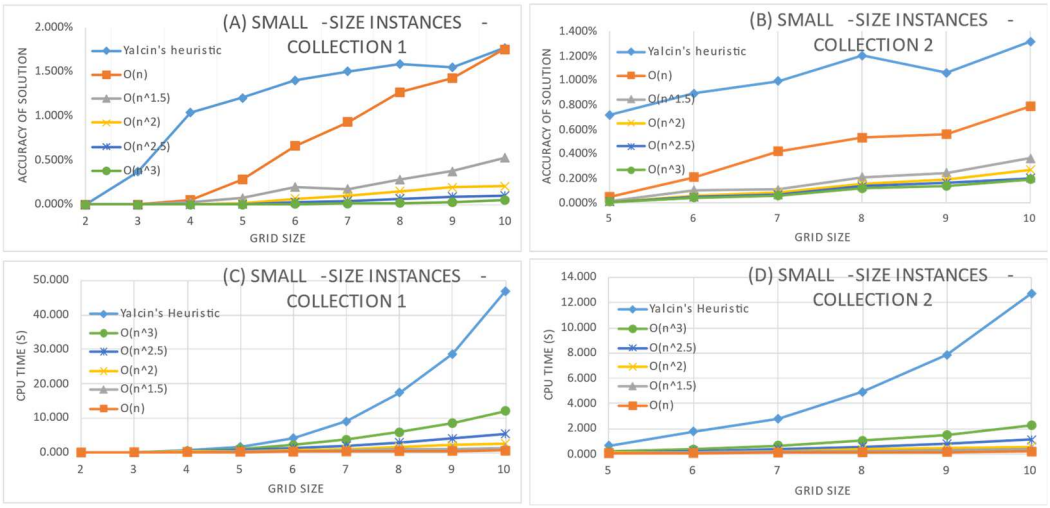

Figure 8. The solution accuracy and CPU time comparison between $beamSearchPBS$ and Yalcin's heuristic

4) Overall comparison of the two heuristics on instances with high storage density

The PBS system is primarily designed to solve high-density storage scenarios. From the perspective of the feasibility of the algorithm, the previous experimental design considered all possible storage densities.

The Gue's research suggests that when the storage density exceeds 90%, PBS systems perform better than the aisle-based storage system in terms of storage or retrieval efficiency (Gue and Kim, 2007). Therefore, to further compare the performance of the two heuristic algorithms in high-density storage scenarios, we choose the high storage density instances with about 10% of escort or less in the two collections. Table 6 provides the computational results of the two heuristics on the small-size instances with high storage density. Please see Table A.5 in Appendices A5 for the features of the groups of the instances considered.

Table 7. Overall solution accuracy comparison of the two heuristics on small-size instances with high storage density

Instance type	G	I	$YalH$	$O(n)$	$O(n^{1.5})$	$O(n^2)$	$O(n^{2.5})$	$O(n^3)$
All groups in collection 1^	157	37164	1.504%	1.057%	0.275%	0.126%	0.059%	0.021%
All groups in collection 2	207	20455	1.031%	0.429%	0.176%	0.130%	0.105%	0.096%
All groups in collections *sum/ #average	364*	57619*	**1.096%#**	**0.570%#**	**0.181%#**	**0.107%#**	**0.072%#**	**0.055%#**
High storage density groups 1^	16	3780	1.274%	0.321%	0.134%	0.055%	0.029%	0.016%
High storage density groups 2	25	2500	0.899%	0.177%	0.081%	0.047%	0.037%	0.036%
High storage density groups *sum /#average	41*	6280*	**1.086%#**	**0.249%#**	**0.108%#**	**0.051%#**	**0.033%#**	**0.026%#**

^Only consider grid size of $5 \times 5 - 10 \times 10$. G: number of groups, I: number of instances, $YalH$: Yalcin's heuristic, $O(n), O(n^{1.5}), \cdots, O(n^3)$: $BeamSearchPBS$ with five computational complexity setting, respectively

Table 6 shows that the average solution accuracy of Yalcin's heuristic in high storage density groups keep almost unchanged compared with the all-groups situation (from 1.096% to 1.086%). However, the average solution accuracy of $BeamSearchPBS$ has more than doubled in 4 out of 5 computational complexity settings. What's more, Table A.6 in Appendices A5 shows that the $BeamSearchPBS$ uses less average computation time while Yalcin's heuristic uses more computation time. Therefore, $BeamSearchPBS$ is more suitable for high-density storage scenarios than Yalcin's heuristic in terms of both algorithm accuracy and algorithm computational time. a deep analysis of this phenomenon on performance differences is described in appendices A5.

5.3.2 Results for large-size instances

In this section, we only consider two representative complexity settings of $BeamSearchPBS$ to compare our heuristic with Yalcin's. Although $BeamSearchPBS$ in all of the complexity settings are better than Yalcin's heuristic in term of solution accuracy when solving the small-size instances, the complexity setting of $O(n)$ deteriorates faster than Yalcin's heuristic with the increase of instance size. Therefore we only consider the complexity settings of $O(n^{1.5})$ and $O(n^3)$ for $BeamSearchPBS$ to do this comparison on the large-size instances, in order to ensure that the overall solution accuracy of $BeamSearchPBS$ is better than that of Yalcin's heuristic, we focus on the improvement of the computational speed and the solution accuracy of $BeamSearchPBS$ relative to Yalcin's heuristic.

1) General results of CPU time

Figure 8 shows the computation time difference of the three algorithms. Due to the huge difference in CPU time consumption of the algorithms, we must use the logarithm of the computation times when we depict them together in a figure (see (A) in Figure 9). The computation times of the three algorithms differ from each other in almost two orders of magnitude.

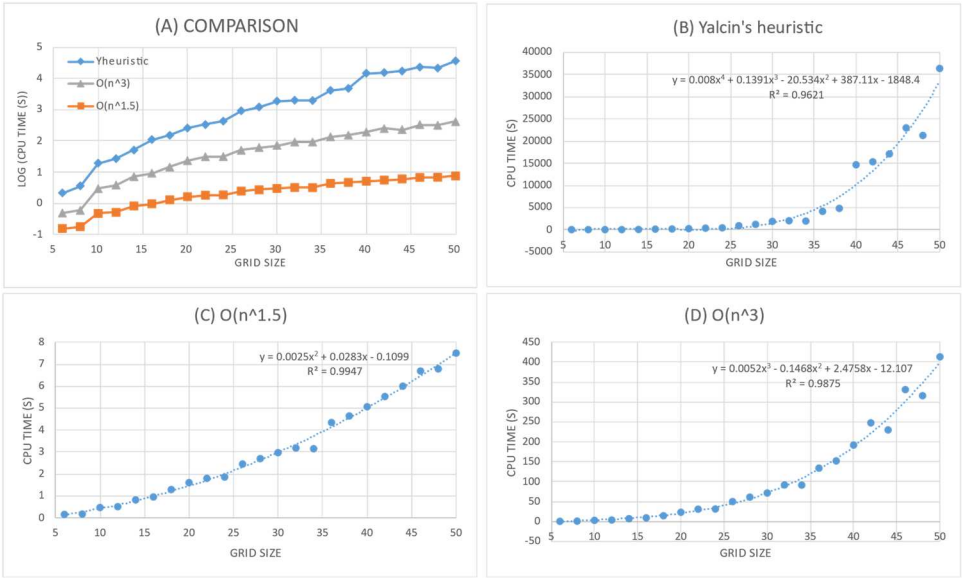

Figure 9. The CPU time comparison between $beamSearchPBS$ and Yalcin's heuristic for large-size instance

2) Function fitting of the computational complexity

Yalcin's heuristic is based on A* algorithm, it is mainly composed of two iterative loops which are the outer R-move loop and inner Escort Estimate loop. It retains only one state at an iteration and refrain from repeatedly visiting a same location, still it may need to traverse all the locations in the grid, therefore, in a PBS system with size $n \times n$, there will be at most n^2 times of R-moves. The inner loop is like the shortest path algorithm, it's computational complexity can be regarded as $O(n^2)$ (Yalcin, 2017), even though the number of states needed to be compared is many times larger (in a constant way) than that in the shortest path algorithm in classical graph theory. Anyway, theoretically the computational complexity of Yalcin's heuristic is $O(n^4)$.

In order to accurately estimate the computational complexity of each algorithm, we fitted the average computational time of different size instances in the experiment by a linear regression model developed as following:

Let $O(n^\gamma)$ be the computational complexity of an algorithm, where γ is the complexity factor we are going to fit by the experimental results, $t_{g,i}$ be the CPU time of instance i in group g, n_g be the grid size of instance group g, I_g be the total number of instances in group g, G be the total number of groups.

The average CPU time of the instances in group g is $t_g = \frac{1}{I_g}\sum_i t_{g,i}$, and let γ_g be the complexity factor of instance group g, let δ be a positive constant factor that makes the equation (17) true.

$$\delta \times n_g^{\gamma_g} = t_g \tag{9}$$

By equation (17), we establish the relationship between grid size and CPU time, from which we can obtain the

complexity factor γ_g by taking the logarithm of equation (17), that i□

$$\log(\delta) + \log(n_g)\gamma_g = \log(t_g) \tag{10}$$

By con□idering $\log(\delta)$ and γ_g a□two variable□and applying the linear regre□ion, we can e□timate the complexity factor γ_g and con□tant factor δ of an algorithm. The following Table 8. give□the e□timation re□ult□.

Table 8. Computational complexity: theoretical and experimental fitting of CPU time

Parameter name	Yalcin's heuristic	$O(n^3)$	$O(n^{1.5})$
δ	0.00043	0.00124	0.00505
Standard deviation of the estimation	0.122	0.073	0.047
γ	4.510	3.236	1.879
Complexity by theoretical analysis	$O(n^4)$	$O(n^3)$	$O(n^{1.5})$
Complexity by fitting experimental data	$O(n^{4.510})$	$O(n^{3.236})$	$O(n^{1.879})$
Theoretical B/Y	---	$1/n$	$1/n^{2.5}$
Fitted B/Y	---	$1/n^{1.274}$	$1/n^{2.632}$

n: grid size $(n \times n)$, B: the CPU time of $BeamSearchPBS$, Y: the CPU time of Yalcin's heuristic.

Although the computational complexity of each algorithm fitted by the experimental re□ult□i□a little higher than that drawn from the theoretical analy□i□, con□idering that thi□increa□e i□□imilar among all the algorithm□, the re□ult□□hould be tru□tworthy in practical application□.

3) Overall comparison

Overall compari□on between the algorithm□i□□hown in Table□9 and 10. From the table□, we can □ee that $BeamSearchPBS$ in the two complexity □etting□are both better than Yalcin'□heuri□tic in term□of □olution accuracy and computation time.

In addition, when □olving the in□tance□of different □ize□and e□cort den□itie□, $BeamSearchPBS$ behave□ more robu□t than Yalcin'□heuri□tic in term□of both CPU time and □olution accuracy. Generally □peaking, the performance difference□between the two algorithm□on large-□ize in□tance□are the □ame in trend and in mechani□m a□tho□e on □mall-□ize in□tance□, □o we will not repeat their analy□i□here. For more detailed information, plea□e refer to the □upplemental data a□□ociated with thi□paper.

Table 9. Overall average move costs and improvement for large-size instances

Algorithms/Improvement	AM cost	GM cost	MAX cost	AM CPU	GM CPU
Yalcin's heuristic	44.46	36.82	136.26	6347.66	722.42
$O(n^3)$	43.74	36.41	133.70	108.97	46.43
$O(n^{1.5})$	44.12	36.65	134.09	3.07	2.15
$O(n^3)$ improvement*	1.622%	1.088%	1.883%	98.283%	93.573%
$O(n^{1.5})$ improvement*	0.751%	0.447%	1.595%	99.952%	99.702%

n: grid size $(n \times n)$, AM: arithmetic mean, GM: geometric mean. All the values in rows 2-4 are the simple average of the corresponding values of all groups, each of which is again the simple average of the corresponding values of all the instances in that

group. Improvement* $= (Y - B)/Y$, where B is the corresponding value of $BeamSearchPBS$, and Y is the corresponding value of Yalcin's heuristic.

Table 10. Overall comparison of $BeamSearchPBS$ with Yalcin's heuristic for large-size instances

Complexity settings	AM cost	AM CPU	# $B < Y$	# $B = Y$	# $B > Y$
$O(n^3)$	1.052%*	47.850%*	28.826%	66.261%	4.913%
$O(n^{1.5})$	0.546%*	90.792%*	24.130%	62.739%	13.130%

n: grid size $(n \times n)$, AM: arithmetic mean, # $B < Y$ means the number of instances of the move cost of $BeamSearchPBS$ is smaller than that of Yalcin's heuristic, # $B = Y$ means the number of instances of the move cost of $BeamSearchPBS$ is equal to that of Yalcin's heuristic, # $B > Y$ means the number of instances of the move cost of $BeamSearchPBS$ is greater than that of Yalcin's heuristic. * calculated by $\frac{100}{G}\sum_{\forall g}\sum_{\forall i}\frac{Y_{g,i}-B_{g,i}}{Y_{g,i}}$, where G is the total number of groups, $B_{g,i}$ and $Y_{g,i}$ are the values of move cost or CPU time of instance i in group g solved by $BeamSearchPBS$ and Yalcin's heuristic, respectively.

6. Conclusions

A beam search algorithm, $BeamSearchPBS$, based on state appraisal and neighborhood search is proposed for minimizing the number of moves of the single-item retrieval problem in a puzzle-based storage system with multiple escorts. $BeamSearchPBS$ can control the computation time and solution accuracy by setting the size of the beam (solution pool) appropriately. Our experiments on a large number of instances show the superiority of $BeamSearchPBS$ over the benchmark methods proposed in the literature, with an improved solution accuracy achieved in less than $1/n^{2.5}$ of their CPU time theoretically, or $1/n^{2.632}$ of their CPU time experimentally. What's more, our heuristic performs even better when dealing with high storage density situation, which is more in line with the primitively defined of PBS systems. By increasing the size of the solution pool, the solution accuracy of $BeamSearchPBS$ can improve further to the limit of about 0.073% which is the estimated opportunity cost of no-backward-move. In addition, the large-size instances of various types of PBS system we generated and their solutions obtained by $BeamSearchPBS$ with different size of the solution pool provide good benchmarks for further theoretical study and commercial applications of PBS.

The improvement in algorithm accuracy (solution quality) has two advantages in practical applications. One is that fewer item moves leads to a reduction in energy consumption. The other is it increases the throughput of a warehouse. Although the accuracy of the Yalcin's algorithm is already very good, our $BeamSearchPBS$ algorithm still achieves an overall improvement of more than 1% over Yalcin's in terms of accuracy, which means the energy consumption and throughput of the warehouse are both improved by over 1%, this is considerable for large e-commerce warehouses. In addition, the improvement on computational efficiency of our algorithm implies reduced computing power requirements, the warehouse can also benefit from both

hardware investment and energy saving in its construction and operation. What is more, our algorithm maybe the only feasible approach for large-scale PBS-based warehouses. We provide five versions of the algorithm with different computational complexities with the overall accuracy and efficiency superior to those of the benchmark algorithm, the managers of those warehouses can thus choose a version of our algorithm with proper complexity by making the best tradeoff between the solution accuracy/system throughput and the computation time.

With the improved efficiency and accuracy, we think the application of $BeamSearchPBS$ in a wider range of warehouses and parking systems is quite promising. For example, the kiva system has been widely adopted by major e-commerce companies around the world, but it was not commonly deployed as a multi-deep warehouse system to achieve higher space efficiency because of its scheduling algorithm constraint. Compared with those SBS/RS that are already very space efficient, PBS can further improve the space utilization, which leads to a shorter distance between items. Consequently, a properly designed PBS system, i.e., a good combination of aisles and PBS modules of different sizes, can not only increase the space utilization, item accessibility, layout and control flexibility, but can also shorten the retrieval time of items. With the development of AGV, shuttle, information system and warehouse design technology, as well as the PBS system, the only bottleneck for implementing a PBS system in practice is an efficient scheduling algorithm. With the aid of $BeamSearchPBS$, a variety of SBS/RS systems may also have a chance to be re-designed as PBS systems and gain the benefits of both space and retrieval efficiency, as well as the flexibility of system design and operation. Possible obstacles to implement a PBS system are the system-specific requirements of its scheduling algorithm in real-world scenarios, which needs a further application-oriented research.

Future research on the application of our algorithm in the operation/optimization of PBS systems may include: 1) improving the setting of solution pool parameters r and p, 2) finding a better state appraisal function through machine learning, 3) relaxing the constraint of no-backward-move, 4) considering multiple-items joint retrieval, 5) finding a better trade-off between retrieval time and energy consumption by allowing block movements, 6) considering the cost of changing the direction of item move, 7) Considering the influence of acceleration and deceleration of shuttles/AGVs, 8) considering the scenario with multiple I/O locations, 9) comparing the performance of PBS systems with other SBS/RS systems, 10) Analyzing the performance of different layout of aisles and PBS modules of different sizes in terms of throughput, 11) designing layout and control policy based on our algorithm.

Fundings

This work was supported in part by the Humanity and Social Science Foundation of Ministry of Education of China (Grant Number 19YJA630054); State Scholarship Fund of China Scholarship Council (Grant Number 201808420328); National Key R&D Program of China (Grant Number 2018YFB1601401); the National Natural Science Foundation of China (Grant Number 71921001); and the Major Program of the National Natural Science Foundation of China (Grant Number 72091210/72091215).

References

Alfieri, A., Cantamessa, M., Monchiero, A., & Montagna, F. (2012). Heuristics for puzzle-based storage systems driven by a limited set of automated guided vehicles. *Journal of Intelligent Manufacturing, 23*(5),1695–1705.

Araya, I., & Riff, M. (2014). A beam search approach to the container loading problem. *Computers and Operations Research*, 43, 100–107.

Araya, I., Moyano, M., & Sanchez, C. (2020). A beam search algorithm for the biobjective container loading problem. *European Journal of Operational Research*, 286, 417–431.

Azadeh, K., De Koster, R., & Roy, D. (2019). Robotized and Automated Warehouse Systems: Review and Recent Developments. *Transportation Science, 53*(4), 917-945. https://doi.org/10.1287/trsc.2018.0873

Birgin, E., Ferreira, J., & Ronconi, D. (2020). A filtered beam search method for the m-machine permutation flow shops scheduling problem minimizing the earliness and tardiness penalties and the waiting time of the jobs. *Computers and Operations Research*, 114, 104824.

Bukchin, Y., & Raviv, T. (2020). Optimal retrieval in puzzle-based storage systems with simultaneous load and block movement. Accessed July 19, 2020, DOI: 10.13140/RG.2.2.29406.05442

Carlo H.J., & Vis, I.F.A. (2012). Sequencing dynamic storage systems with multiple lifts and shuttles. *International Journal of Production Economics*, 140(2), 844-853.

Dijkstra, E. W. (1959). A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik* 1 (1): 269–271.

Doppler. (2016). "DCS 2DL." Accessed December 7, 2019. http://doppler.gr/en/products/parking-systems/dps-xryc/

Ekren, B.Y., Akpunar, A., Sari, Z., & Lerher, T. (2018). A tool for time, variance and energy related performance estimations in a shuttle-based storage and retrieval system. *Applied Mathematical Modelling*. 63, 109-127.

Fragapane, G., De Koster, R., Sgarbossa, F., & Strandhagen, J.O.(2021). Planning and control of autonomous mobile robots for intralogistics: Literature review and research agenda, *European Journal of Operational Research (2021)*, doi: https://doi.org/10.1016/j.ejor.2021.01.019

Furmans, K., Nobbe, C., Schwab, M. (2011). Future of material handling–modular, flexible and efficient. Amato, N.M., Chen, I.M., De Luca, A., Jenkins, C., Kragic, D., Papanikolopoulos, N., Park, F., Parker, L.,

Sugano, S., van der Stappen, F., eds. *IEEE/RSJ Internat. Conf. Intelligent Robots Systems (Institute of Electrical and Electronics Engineers, Piscataway, NJ).*

Gue, K.R., & Kim, B. S. (2007). Puzzle-based storage systems. *Naval Research Logistics(NRL),* 54(5), 556–567.

Gue, K.R., Furmans, K., Seibold, Z., & Uludag, O. (2014). GridStore: A Puzzle-Based Storage System With Decentralized Control. *IEEE Transactions on Automation Science and Engineering 11* (2), 429–438.

Gebhardt. (2016). "GEBHARDTFlexConveyor." Accessed December 7, 2019. http://www.gebhardt-foerdertechnik.de/de/produkte/flexconveyor/

ICAM. (2016). "SMOOV ASRV GENERATION 2." Accessed December 7, 2019. http://www.smoov-asrv.eu/

IFL, Institut für Fördertechnik und Logistiksysteme. 2015. "GridStore." Accessed December 7, 2019. https://www.youtube.com/watch?v=LxzBuB_JCN4

Jerman, B., Ekren, B. Y., Küçükyaşar, M., & Lerher, T. (2021). Simulation-Based Performance Analysis for a Novel AVS/RS Technology with Movable Lifts. *Applied Sciences*, 11(5), 2283.

Kota, V.R., Taylor, D., & Gue, K.R. (2010). Retrieval Time Performance in Puzzle-based Storage Systems. *In Proceedings of the 2010 Industrial Engineering Research Conference (IERC)*, edited by A. Johnson and J. Miller, 1558–1563. Cancun, Mexico.

Kota, V.R., Taylor, D., & Gue, K.R. (2015). Retrieval Time Performance in Puzzle-based Storage Systems. *Journal of Manufacturing Technology Management 26* (4): 582–602.

Kress, D., Dornseifer, J., & Jaehn, F. (2019). An exact solution approach for scheduling cooperative gantry cranes. *European Journal of Operational Research*, 273, 82–101.

Küçükyaşar, M., Ekren, B. Y., & Lerher, T. (2021). Cost and performance comparison for tier-captive and tier-to-tier SBS/RS warehouse configurations. *International transactions in operational research*, 28, 1847-1863.

Lerher, T. (2018). Aisle changing shuttle carriers in autonomous vehicle storage and retrieval systems. *International Journal of Production Research,* 56(11), 3859-3879.

Lerher, T., Borovinšek, M., Ficko, M., & Palčič, I. (2017). Parametric study of throughput performance in SBS/RS based on simulation. *International journal of simulation modelling*, 16(1), 96-107.

Lerher, T., Ficko, M., & Palčič. I. (2020). Throughput performance analysis of automated vehicle storage and retrieval systems with multiple-tier shuttle vehicles. *Applied mathematical modelling*, 2020 doi: https://doi.org/10.1016/j.apm.2020.10.032

Li, Z., Kucukkoc, I.,& Tang, Q. (2021). Enhanced branch-bound-remember and iterative beam search algorithms for type II assembly line balancing problem. *Computers and Operations Research*, 131, 105235.

Marchet, G., Melacini, M., Perotti, S., &Tappia, E. (2012). Analytical model to estimate performances of autonomous vehicle storage and retrieval systems for product totes. *International Journal of Production Research*, 50(24), 7134–7148.

Mayer, S.H. (2009). Development of a Completely Decentralized Control System for Modular Continuous Conveyors. *PhD diss. Karlsruhe. Karlsruhe, Univ., Diss., 2009*. http://d-nb.info/101409898X/34.

Mirzaei, M., De Koster, R., & Zaerpour, N. (2017). Modelling load retrievals in puzzle-based storage systems. *International Journal of Production Research. 55*(21):6423–6435.

Mutrade. (2014). Automatic Parking System. Accessed December 7, 2019. http://www.mutrad.com/AutomaticParkingSystem.html

ODTH. (2015). Magic Black Box. Accessed December 7, 2019. https://www.odth.be/black-box/

Parreño, F., Alonso, M. T., & Alvarez-Valdes, R. (2020). Solving a large cutting problem in the glass

manufacturing industry. *European Journal of Operational Research*, 287, 378–388.

Ratner, D,, Warmuth, M. (1986). Finding a Shortest Solution for the N × N Extension of the 15-PUZZLE is Intractable. In AAAI, edited by Tom Kehler, 168–172. Burlington, MA: Morgan Kaufmann.

RR Parkon. (2014). Puzzle Parking. Accessed December 7, 2019. http://www.rrparkon.com/product/puzzle-parking

Woehr. (2016). Liverpool – Parksafe 583. Accessed December 7, 2019. http://www.woehr.de/en/project/liverpool-parksafe-583.html

Yalcin, A. (2017). Multi-Agent Route Planning in Grid-Based Storage Systems. Doctoral dissertation of der Europa-University Viadrina in Frankfurt.

Yalcin, A., Koberstein, A., & Schocke, K. O. (2019a). 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, 57*, 1, 143-165, DOI: 10.1080/00207543.2018.1461952.

Yalcin, A., Koberstein, A., & Schocke, K. O. (2019b). Motion and layout planning in a grid-based early baggage storage system: Heuristic algorithms and a simulation study. *OR Spectrum* 41, 683-725.

Yu, Y., Yu, H., De Koster, R., & Zaerpour, N. (2019). Optimal algorithm for minimizing retrieval time in puzzle-based storage system with multiple simultaneously movable empty cells. Working paper, School of Management, University of Science and Technology of China, Hefei.

Zaerpour, N., Yu, Y., & De Koster, R. (2015). Small is Beautiful: A Framework for Evaluating and Optimizing Live-Cube Compact Storage Systems. *Transportation Science. 51*, 34–51. doi:10.1287/trsc.2015.0586.

Zaerpour, N., Yu, Y., & De Koster, R. (2017a). Optimal two-class-based storage in a live-cube compact storage system. *IISE Trans. 49*(7), 653–668.

Zaerpour, N., Yu, Y., & De Koster, R. (2017b). Response time analysis of a live-cube compact storage system with two storage classes. *IISE Trans. 49*(5), 461–480.