Hindawi
Discrete Dynamics in Nature and Society
Volume 2021, Article ID 6646180, 16 pages
https://doi.org/10.1155/2021/6646180

Research Article

Large-Scale Storage/Retrieval Requests Sorting Algorithm for Multi-I/O Depots Automated Storage/Retrieval Systems

Yu Bo Song[1] **and Hai Bo Mu**[2]

[1]*Institute of Mechatronic Technology, Lanzhou Jiaotong University, Lanzhou 730070, China*
[2]*School of Traffic and Transportation Engineering, Lanzhou Jiaotong University, Lanzhou 730070, China*

Correspondence should be addressed to Yu Bo Song; songyubo@mail.lzjtu.cn

Received 3 October 2020; Revised 17 January 2021; Accepted 4 May 2021; Published 20 May 2021

Academic Editor: Luca Pancioni

This paper addresses the sequence sorting problem of large-scale storage/retrieval (S/R) requests in multiple Input/Output (multi-I/O) depots automated storage/retrieval systems (AS/RS), in which the cargoes can enter/leave the system through multi-I/O depots, the stacker can load only one cargo, and the load travel time of stacker is fixed. The problem is to find an optimal sequence for a certain S/R requests sequence, and it is a special kind of traveling salesman problem. In this paper, a heuristic algorithm based on assignment is proposed. In order to eliminate the subloops emerged in the sorting process, the equivalent merging and minimum cost merging methods of subloops are considered, and the proposed algorithm is modified. Experimental results indicate the effectiveness of the proposed algorithm.

1. Introduction

The functions of automated storage/retrieval system (AS/RS) include receiving, storing, and order fulfillment. Assignment of products to storage locations, choice of products to retrieval, and storage/retrieval (S/R) requests sequencing are important planning and operational decisions in the AS/RS operation problems [1, 2]. Since AS/RS is required to accomplish S/R requests in a timely manner, optimization of S/R requests sequence of a stacker is considered as an essential problem for efficient operation in the AS/RS [3, 4]. Especially, the optimization of S/R requests sequence becomes much more important for AS/RS, which is fully integrated with other automated industrial production line or logistical systems. Not only the optimized routing, but also the fast calculation of routing is required to be attained simultaneously. With the increasing scale of S/R requests sequence, the scale of its feasible region increases exponentially. The overall performance and efficiency of AS/RS will be limited if the optimization of routing takes considerable time [5–7].

However, it is challenging to quickly optimize the execution order of large-scale S/R requests sequence. Research has gained momentum with the papers by Hausman et al. [8] and Graves et al. [9]. A lot of optimization methods of S/R requests sequence under different AS/RS layout structure are proposed. In AS/RS with single Input/Output (I/O), the optimization methods of S/R requests sequence are distributed in three operation modes, namely, single command cycle, double command cycle, and multiple command cycle. In single command cycle mode, the time to complete all S/R requests is a fixed value, and S/R requests sorting is not an optimization problem. However, after importing the permission of S/R requests, the S/R requests sequence sorting problem is transformed into a polynomial solvable problem and NP-hard problem under different optimization objectives [10, 11]. In double command cycle mode, a stacker can complete one storage request and one retrieval request in the same command cycle. Based on the feature that the S/R requests appear in pairs, Lee and Schaefer [12] constructed a bipartite assignment graph by using storage request and retrieval request as the job request nodes, respectively. Hungarian algorithm was used to pair the S/R requests in the

same command cycle, and the completion time of S/R requests was minimized. Compared with single command cycle, completing the same number of S/R requests in the double command cycle mode can save about 30% of the time cost [9, 13]. But it is not always possible to have pairs of S/R requests to construct dual instruction cycles in the actual operation process. Eben-Chaime and Pliskin put forward the mixed operation mode and made the decision rules for selecting single or double command cycle [14, 15].

Due to the excellent efficiency of dual command cycle, researchers proposed the concept of expanding the number of shuttle in the stacker and thus resulted in four command cycles (twin-shuttle stacker) and six command cycles (triple-shuttle stacker). Keserla and Peters [16] proposed a minimum perimeter based heuristic algorithm and realized the minimization of S/R requests operation time in a four-command cycle. To solve the same problem, Sarker et al. proposed some optimization methods based on greedy strategy to improve the throughput of AS/RS [17, 18]. Tanaka and Araki [19] established a capacitated stacker S/R requests sorting model for multishuttle AS/RS under the condition of fixed storage and retrieval locations and designed an exact algorithm to solve the model. Tanaka [20] extended their work and proposed a hybrid tabu heuristic algorithm to solve the S/R requests sequence of stacker. Popovic et al. [21] provided a general framework for the optimization of S/R requests sequence in the operation mode of a six command cycle, and heuristic algorithm and genetic algorithm were used separately to optimize the S/R requests sequence. They believed that the optimization effect of genetic algorithm was better than that of the heuristic algorithm for the optimization of S/R requests sequence in multi-command cycle. Based on shared memory strategy, Tanaka and Araki [22] and Yang et al. [23] analyzed the optimization problem of S/R requests sequence in multi-command cycle and formalized the problem into an integer programming model. Accurate algorithm and variable neighborhood search algorithm were used to solve the presented model, respectively.

The optimization methods of S/R requests sequence of the above three operation modes are discussed in single I/O depot AS/RS, in which the I/O depot is located at one end of the aisle, and each S/R request has the characteristic of periodic return to the unique I/O depot. Another common layout structure of AS/RS is that I/O depot is located at both ends of the aisle. The expansion of I/O depot increases the paths for cargoes' entering or leaving the AS/RS and also provides the possibility of separating the storage and retrieval operations. Pohl et al. consider that the decisions including AS/RS layout and aisle configuration have an impact on the efficiency of AS/RS [24]. The recent work of Schleyer and Gue [25] and Gue et al. [26] also revealed this conclusion mentioned above. Gharehgozli et al. [27] optimized the S/R requests sequence through polynomial algorithm under the following three conditions: (i) the output depot was dedicated; (ii) the input depot and the output depot were separated; and (iii) the stacker could start and end at any position, and the method was compared with the optimization method presented by Van den Berg and Gademann [28]. Researchers can refer to the summary of Roodbergen and Vis [4], and Boysen and Stephan [29] to obtain more research results related to AS/RS.

The literatures mentioned above studied the optimization of S/R requests sequence for AS/RS with single and double I/O depots, and the optimization methods are all on the basis of command cycle. To the best of our knowledge, few people pay attention to the optimization of S/R requests sequence for AS/RS with no command cycle division. In this paper, the optimization problem of large-scale S/R requests sequence for AS/RS with multi-I/O depots is studied based on the logistics problem of actual air cargo terminal. The AS/RS with multi-I/O depots is composed of two rows of racks, and the I/O depot is distributed along both sides of the aisle. Each I/O depot, through which the cargo enters or leaves the system, is connected to the conveyor (the layout of AS/RS with multi-I/O depots is shown in Figure 1). The stacker is located in the aisle, and its vertical lifting structure and horizontal moving structure can run simultaneously to complete S/R request of cargoes. A stacker can only hold one cargo during its operation process. Then, the optimization problem of large-scale S/R request sequence in multi-I/O depots AS/RS is to find an optimal execution sequence for a given group of S/R requests, with the purpose of minimizing the operation time of completing all S/R requests. In order to solve the problem mentioned above, we propose a fast optimization method in this paper to study the optimization problem of S/R requests sequence, especially for large-scale S/R requests sequence, in multi-I/O depots AS/RS under the condition of no command cycle division. Simulation results indicate that the algorithm is superior in the quality of optimization results and calculation time.

The paper is organized as follows. Details of the optimization problem of S/R requests sequence in multi-I/O depot AS/RS are introduced in Section 2. Subsequently, an assignment based heuristic algorithm to optimize large-scale S/R requests sequence is presented in Section 3. Computer simulation results and conclusions follow in Sections 4 and 5.

2. The R/S Requests Sequence Problem

According to the operation process of S/R requests, we divide the execution process of S/R requests into two consecutive steps. For the storage request, first, the stacker moves from the current position to the I/O depot and loads the cargo, and then the stacker moves with load to the storage location of the cargo and unloads. For the retrieval request, first, the stacker moves without load from the current position to the storage location of retrieval cargo and loads the cargo, and then the stacker moves with load to the output depot and unloads the cargo. The trajectory formed by the continuous execution of S/R requests of stacker is called a route. A route may be seen as a concatenation of loaded travels, with no-load travels in-between. If the end position of a load travel is the start position of another load travel, the no-load travel is a virtual travel, and its running time is 0. A route for the stacker to complete all S/R requests corresponds to an S/R requests sequence.

Through the whole study, we considered the following assumptions. Firstly, only a group of static S/R requests is considered, which means that the scale of the S/R requests sequence remains unchanged during the whole optimization process. Secondly, the Chebyshev distance is used to represent the movement time of the stacker between any two points. Here, the Chebyshev distance refers to the maximum travel time between vertical travel time and horizontal travel time. Thirdly, the stacker moves both vertically and horizontally simultaneously at a constant speed. Fourthly, each location can store only one cargo. Finally, the initial position of the stacker is the virtual starting and ending position for the problem.

We allow the cargo to enter/leave the system through any I/O depot and attempt to find an optimal or near optimal execution sequence of the stacker under the condition that the I/O depot of S/R requests and storage position are all known. It is convenient for us to define the following symbols for problem modeling.

m: number of storage requests

n: number of retrieval requests

S_i: the ith storage request

R_j: the jth retrieval request

SI_i: input depot of the ith storage request

SL_i: storage position of the ith storage request

RO_j: retrieval depot of the jth retrieval request

RL_j: retrieval position of the jth retrieval request

L_k: the kth load travel

t_{kl}: the no-load movement time between load travel L_k and L_l

SP: start position of stacker

EP: end position of stacker

$t(A, B)$: movement time of stacker from position A to position B

For each storage request S_i, a load travel ($\mathrm{SI}_i \longrightarrow \mathrm{SL}_i$) must be performed, $i = 1, \ldots, m$. Likewise, for each retrieval request R_j, a load travel ($\mathrm{RL}_j \longrightarrow \mathrm{RO}_j$) must be performed, $j = 1, \ldots, n$. The no-load travel is executed after each load travel, and its travel time is the time from the end of the previous load travel to the start of the next load travel. We want to find a sequence of load travel to minimize the completion time of the S/R requests sequence. Figure 2 depicts such a sequence. The stacker starts from the initial position SP and returns to the initial position SP after completing all S/R requests. The previous and subsequent load travels of any load travel can be selected arbitrarily, and any two load travels are connected by a no-load travel. Different connection scheme of load travel corresponds to different operation path and completion time of stacker. Since the time of load travel is fixed, the completion time of S/R requests sequence is decided by the running time of no-load travel, and the optimal sequence of no-load travel corresponds to the optimal execution sequence of S/R requests. We consider each load travel as an S/R request vertex and denote it by $L_p\,(p = 1, \ldots, m+n)$. Each vertex is connected by a no-load travel $E_q\,(q = 1, \ldots, m+n)$, and the minimization of S/R requests completion time is transformed into the minimization of time for traversing all the S/R requests vertices. In the process of traversing, each S/R request vertex can be traversed for only one time, and the stacker finally returns to the initial position. The optimal solution of traveling salesman problem (TSP) is exactly the optimal scheme of no-load travel selection and sorting. Therefore, we model the optimization problem of S/R requests sequence as a TSP.

It is assumed, in this model, that there are m storage requests and n retrieval requests, and the starting address and the ending location of the stacker are both SP. The decision variable x_{kl} takes on binary values, 0 or 1, depending on whether or not the load travel L_l is performed immediately after the load travel L_k has been performed by the stacker. Let t_{kl} be the no-load movement time between load travel L_k and L_l. The objective function minimizes the total no-load travel time. Constraints (2) and (3) ensure that each load travel is accessed once and only once. Constraint (4) ensures that, in the optimal solution, no-load travel and load travel do not constitute a subloop.

$$\min T = \sum_{l=1}^{m+n+1} \sum_{k=1}^{m+n+1} t_{kl} x_{kl}, \quad k \neq l, \tag{1}$$

$$\sum_{k=1}^{m+n+1} x_{kl} = 1, \quad \forall l \in L \cup \{\mathrm{SP}\}, l \neq k, \tag{2}$$

$$\sum_{l=1}^{m+n+1} x_{kl} = 1, \quad \forall k \in L \cup \{\mathrm{SP}\}, k \neq l, \tag{3}$$

$$\sum_{l \in L'} \sum_{k \in L'} x_{kl} \leq \left|L'\right| - 1, \quad \forall L' \subset L \cup \{\mathrm{SP}\}, 2 \leq \left|L'\right| \leq m+n, \tag{4}$$

$$x_{kl} \in \{0, 1\}. \tag{5}$$

In order to quickly solve the optimization problem of large-scale S/R requests sequence rapidly, we develop an assignment based heuristic algorithm, which will be introduced in the next section.

3. Heuristic Algorithms for the R/S Requests Sequence Problem

In this section, a heuristic method to solve the optimization problem of large-scale S/R requests sequence in multi-I/O depots AS/RS was presented. The presented heuristic algorithm is based on the solution idea of assignment model, and it can optimize large-scale S/R requests sequence in a near-real-time manner. It may be a useful tool for dynamic optimization of S/R requests sequence.

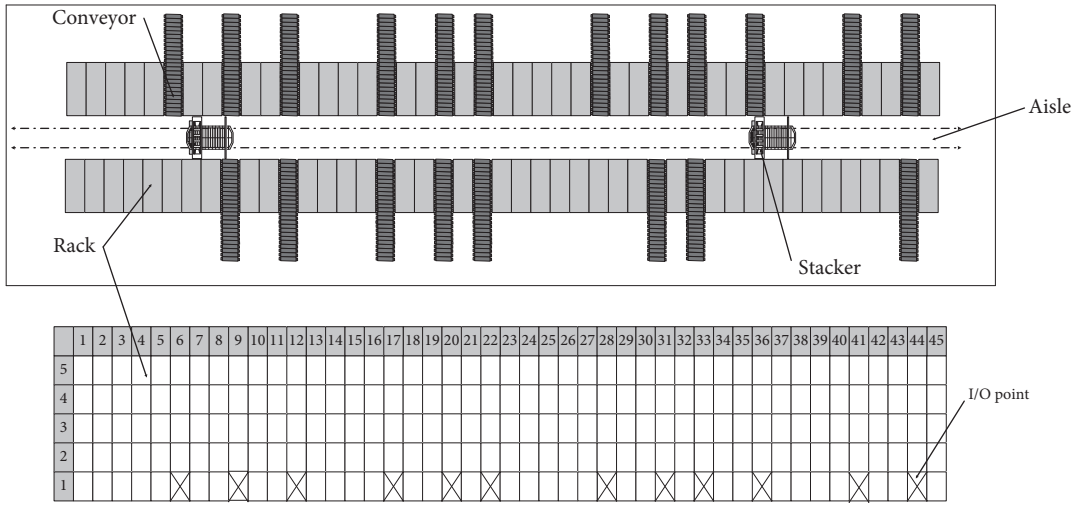

Figure 1: Layout of multi-I/O depots AS/RS.

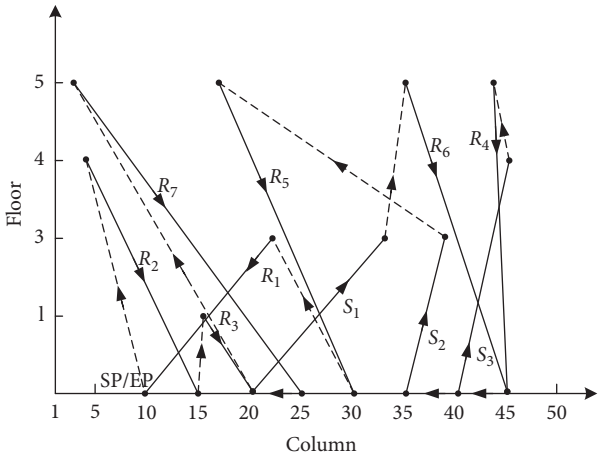

Figure 2: Stacker movement route with three storage requests and seven retrieval requests: the horizontal axis and the vertical axis represent the column and floor, where the stacker access point is located, respectively, the solid line represents load travel, and the dotted line represents no-load travel.

3.1. Algorithmic Idea. According to the TSP model built in Section 2, each node in the model represents an S/R request. Under the condition that the start location and destination location of each S/R request are known, if the initial point returned by the stacker is assumed to be EP, then each vertex in the TSP model can be divided into two vertices, and the assignment problem (AP) we constructed is as follows. The constructed bipartite graph is represented as $G = (V_1, V_2, E)$. V_1 is composed of the start points of $m + n + 1$ no-load travels in the tour including the end point of the storage requests $(\mathrm{SL}_1, \ldots, \mathrm{SL}_m)$, the end point of the retrieval requests $(\mathrm{RO}_1, \ldots, \mathrm{RO}_n)$, and the start point of the stacker (SP), and V_1 is defined as the set of people in the AP. V_2 is composed of the end points of $m + n + 1$ no-load travels in the tour, including the start point of the storage request $(\mathrm{SI}_1, \ldots \mathrm{SI}_m)$, the start point of the retrieval request $(\mathrm{RL}_1, \ldots \mathrm{RL}_n)$, and the end point of the stacker (EP), and V_2 is defined as the set of tasks in the AP. The set E is composed of edges between any vertex in V_1 and any vertex in V_2. The cost of any edge is equal to the no-load travel time. E is defined as the time for each person to complete each task in the AP. The time of no-load travel is defined in Table 1. The edge with value ∞ represents the prohibited travel, and prohibited travel is omitted in bipartite graph.

In addition, each task in the AP can only be completed by one person, and the constraint that each person can only undertake one task is equivalent to constraints (2) and (3) in the TSP model. Then, the minimum edge cover of bipartite graph is the assignment scheme of AP.

According to the above rules, we construct a bipartite graph with two storage requests and two retrieval requests (see Figure 3) and arbitrarily give an assignment scheme as shown in Figure 4(a). According to the reverse rule of constructing bipartite graph, the vertices in V_1 and V_2 of Figure 4(a) are combined into the S/R requests node and the start point of the stacker. According to the definition of edge in Table 1, there is no edge between the start location and the destination location of the same S/R request and between SP and EP. Therefore, the edges in the assignment scheme are not lost in the process of vertex merging. Since the assignment scheme covers all the vertices in V_1 and V_2, and each vertex in V_1 and V_2 has only one edge connected, each merged S/R request node has two edges connected as shown in Figure 4(b). By merging the same edges in Figure 4(b), we can get a node coverage scheme with multiple disjoint subloops, and the covered nodes include all the S/R request nodes and the start point of the stacker as shown in Figure 4(c). The set of edges in Figure 4(c) is exactly the same as that in Figure 4(a). Therefore, a node traversal scheme with multiple disjoint subloops of TSP can be found according to the above method, and vice versa. In conclusion, our AP has the same solution space as the TSP with subloop relaxation.

Based on the above analysis, although the optimal assignment scheme, which is a node coverage scheme with multiple disjoint subloops, may not satisfy constraint (4), if we can find the combination scheme of subloops without

Table 1: Costs related to edges in the sequencing problem.

From	To	Cost
SP	SI_i	$t(SP, SI_i), i = 1, \ldots, m$
SP	RL_j	$t(SP, RL_j), j = 1, \ldots, n$
SP	EP	∞
$SL_{i'}$	SI_i	$t(SL_{i'}, SI_i), i = 1, \ldots, m, i' = 1, \ldots, m$, and $i \neq i'$
$SL_{i'}$	RL_j	$t(SL_{i'}, RL_j) i' = 1, \ldots, m, j = 1, \ldots, n$
$SL_{i'}$	SI_i	$\infty, i = 1, \ldots, m, i' = 1, \ldots, m$ and $i = i'$
$SL_{i'}$	EP	$t(SL_{i'}, EP), i' = 1, \ldots, m$
$RO_{j'}$	SI_i	$t(RO_{j'}, SI_i), i = 1, \ldots, m, j' = 1, \ldots, n$
$RO_{j'}$	RL_j	$t(RO_{j'}, RL_j), j = 1, \ldots, n, j' = 1, \ldots, n$ and $j \neq j'$
$RO_{j'}$	RL_j	$\infty, j = 1, \ldots, n, j' = 1, \ldots, n$ and $j = j'$
$RO_{j'}$	EP	$t(RO_{j'}, EP), j' = 1, \ldots, n$

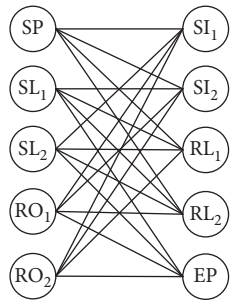

Figure 3: Bipartite graph with two storage requests and two retrieval requests.

increasing the no-load travel time, we can obtain the optimal solution of TSP problem. Due to the same specifications of the shelf in AS/RS, and the constant running speed of the stacker in the horizontal and vertical directions, with the increase of the size of the S/R requests sequence, there is a great probability that an edge with the same no-load running time exists in different subloops. Therefore, in the large-scale S/R requests sequence, there is a great probability that an equivalent merging scheme of subloops exists. In order to show the association between the optimal solution of the AP and the optimal solution of the TSP, we give the following theory for three cases: there is no subloop, there is equivalent merging subloop, and there is no equivalent merging subloop.

Definition 1. The scale of TSP refers to the number of load travels plus 1 in the optimization problem of S/R requests sequence.

Definition 2. The scale of AP refers to the number of people or tasks.

Definition 3. In an AP, assigning a task to a person is called an assignment pair, and in the optimal solution of an AP, the number of assignment pairs is equal to the scale of the AP.

Theorem 1. *If the scales of the TSP and the AP are equal, then the number of edges in the optimal solution of the TSP is equal to the number of assignment pairs of the AP.*

Proof. It can be concluded from Definition 1 that there are N-1load travels for the TSP with size N. Departing from the start point of the stacker, one edge is added before the first load travel is performed, and N-2 edges will be generated after the remaining load travels have been completed. Finally, another edge is needed when returning to the initial point from the position, where the loading travel is completed. Then, all feasible solutions of TSP contain N edges. Since the feasible solution contains the optimal solution, there are also N edges in the optimal solution. According to Definition 3, the number of assignment pairs is N in the optimal solution of the AP with size N. Therefore, Theorem 1 holds. □

Corollary 1. *For an AP and a TSP with the same scale, under the condition that one person only completes one task, the optimal solution of the AP corresponds to the optimal solution of the TSP with the same scale if the no-load travels that correspond to the assignment pairs in the optimal solution of AP and the load travels that correspond to the S/R requests sequence do not constitute a subloop.*

Proof. It can be concluded from Definition 3 that an assignment pair corresponds to a no-load travel, and the optimal solution of AP is the combination scheme with the minimum total time among all the combination schemes of no-load travels. One person completes only one task means that the start point and end point of each no-load travel are connected with different load travels and connected with only one load travel. It can be concluded from Theorem 1 that the numbers of no-load travels in the optimal solution of AP and TSP with the same scale are equal. If the no-load travels that correspond to the assignment pairs in the optimal solution and the load travels do not constitute a subloop, the N assignment pairs in the optimal solution of AP with scale N can connect the N load travels in series successively to form a path that begins and ends at the start location and ensure that each vertex is visited only once. Therefore, the optimal solution of AP is also the optimal solution of TSP. □

Corollary 2. *If the load travels corresponding to the S/R requests and the no-load travels corresponding to the assignment pairs in the optimal solution of AP constitute at least two subloops, and there exists a subloop merging scheme, which does not change the task completion time of the AP, then the optimal solution of AP is also the optimal solution of TSP.*

Proof. If the load travels corresponding to the S/R requests and the no-load travels corresponding to the assignment pairs in the optimal solution of AP constitute at least two subloops, then load travels and no-load travels will constitute two types of paths. One is the subloop containing the start and end points of the stacker and another is the subloop that does not contain the start and end points of the stacker. Figure 5(a) shows the subloop SC_1 that contains the start and end points of the stacker and the subloop SC_2 that does not contain the start and end points of the stacker. In

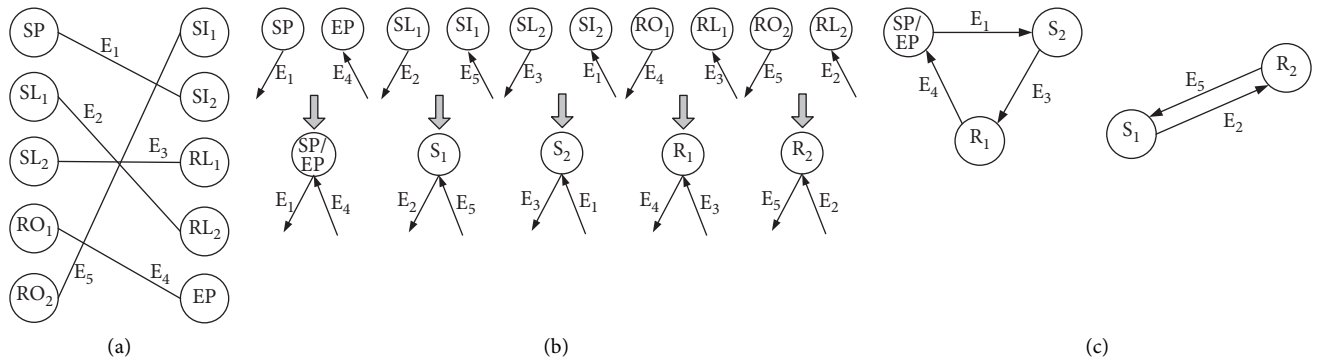

Figure 4: Solution of TSP with subloop relaxation transformed from assignment scheme. (a) Assignment scheme. (b) Node merging. (c) Solution of TSP problem with subloop relaxation.

Figure 5(b), no-load travel E_k that connects $A \in \{\mathrm{EP}, \mathrm{SL}_1, \ldots \mathrm{SL}_m, \mathrm{RO}_1, \ldots \mathrm{RO}_n\}$ and $B \in \{\mathrm{SP}, \mathrm{SI}_1, \ldots \mathrm{SI}_m, \mathrm{RL}_1, \ldots \mathrm{RL}_n\}$ and no-load travel E_l that connects $C \in \{\mathrm{SP}, \mathrm{SI}_1, \ldots \mathrm{SI}_m, \mathrm{RL}_1, \ldots \mathrm{RL}_n\}$ and $D \in \{\mathrm{EP}, \mathrm{SL}_1, \ldots \mathrm{SL}_m, \mathrm{RO}_1, \ldots \mathrm{RO}_n\}$ are deleted, while no-load travel E_k' that connects A and C and no-load travel E_l' that connects B and D are added. A new tour is formed by the above operation, as shown in Figure 5(c). If the sum of travel times of no-load travels E_k and E_l is equal to the sum of travel times of no-load travels E_k' and E_l', then the sum of no-load travel time $t(A, C) + t(D, B)$ in the tour of Figure 5(c) is equal to the sum of no-load travel time $t(A, B) + t(D, C)$ in Figure 5(a). The other no-load travel times in SC_1 and SC_2 remain unchanged. Therefore, when the subloops are merged, the sum of the no-load travel time of the new tour after merging is equal to the sum of the no-load travel time of subloop SC_1 and SC_2. The merger of more subloops without the start and end points of the stacker can be proved in the same way. According to the equivalent merging process of subloops, all of the load travel vertices involved in the subloop merging process can be accessed only once, and the time does not change. Therefore, the optimal solution of AP formed by merging subloops is also the optimal solution of TSP. □

Corollary 3. *If the load travels corresponding to the S/R requests and the no-load travels corresponding to the optimal solution of the AP constitute at least two subloops, and there is no subloop merging scheme, which does not change the task completion time of the AP, then the subloop merging scheme with the smallest task completion time increment is selected to merge the subloops one by one until all subloops are merged, and the solution of AP after subloop merging is also the optimal solution or suboptimal solution of TSP.*

Proof. According to the proof of Corollary 2, the merging of subloops needs to delete two existing no-load travels and generate two new no-load travels at the same time. When there is no equivalent merging scheme, the no-load travel time will be increased after the merging of subloops. The increase of no-load travel time will bring two possibilities. One is that the no-load travel time of the merged loop is equal to the time of the optimal solution of TSP (as shown in Table 2). The total time of no-load travels corresponding to the optimal solution of the AP, which is constituted by the start and end points of the seven load travels in Table 2 and the start and end points of the stacker, is 162.8 seconds, and three subloops are formed (see Figure 6(a)). The three subloops are merged twice according to the minimum cost rule and form a tour (see Figure 6(b) and Figure 6(e)), and the no-load travel time is increased by 4.6 seconds and 25.8 seconds, respectively. The sum of no-load travel time of the tour is 193.2 seconds, which is equal to the time of the optimal solution of TSP obtained by enumeration method. Another is that the no-load travel time of the merged loop is larger than the time of the optimal solution of TSP (as shown in Table 3). The total time of no-load travels corresponding to the optimal solution of the AP, which is constituted by the start and end points of the nine load travels in Table 3 and the start and end points of the stacker, is 231.2 seconds, and three subloops are formed (see Figure 7(a)). The three subloops are merged by equivalent and minimum cost rules and form a tour (see Figures 7(b) and 7(e)). The minimum cost merging increases the no-load travel time by 10 seconds, and the sum of no-load travel time of the tour is 241.2 seconds, which is 1.4 seconds longer than the time of the optimal solution of TSP obtained by enumeration method. Therefore, the optimal or suboptimal solution of TSP can be found by merging the minimum cost subloops. □

3.2. Algorithmic Design. The stacker starts from the start point SP and returns to the start point after completing all S/R requests. We mark the end point as a virtual point EP. Based on Corollary 1, we propose a heuristic algorithm based on assignment. The algorithm consists of five steps: constructing bipartite graph, creating adjacency matrix, solving the optimal assignment scheme, determining the load travel corresponding to the vertex of serial S/R requests, and sorting the S/R requests sequence.

Figure 8 shows an example with four storage requests and three retrieval requests. The start and end points of each S/R request are the start and end points of load travel. First, we construct the bipartite graph corresponding to Figure 8 according to the bipartite graph defined in section 3.1 (see Figure 9).

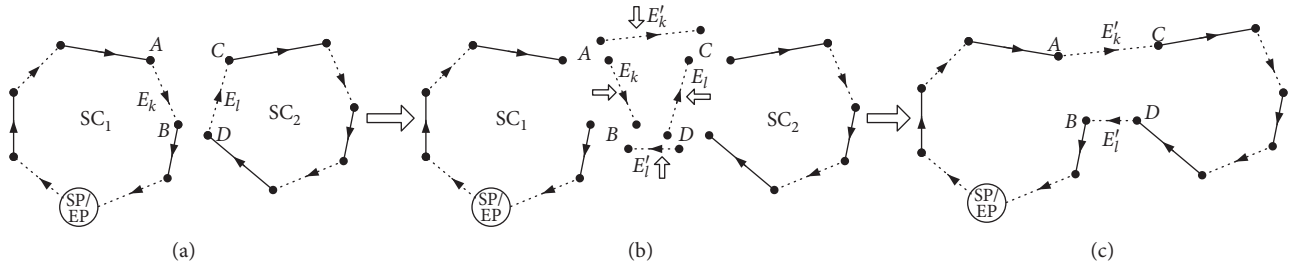

Figure 5: Process diagram of subloops merging. (a) Examples with subloops. (b) Merging scheme of subloops. (c) Loop formed with merging.

Table 2: An example of the optimal solution of TSP obtained by minimum cost merging.

Number	Load travels				No-load travels determined by assignment	Subloops
	Start point		End point			
	Column	Layer	Column	Layer		
SP/EP	10	1	—	—	SP ⟶ 5	SC1
1	31	3	40	1	5 ⟶ EP	SC1
2	48	4	50	1	1 ⟶ 6	SC2
3	30	1	38	3	6 ⟶ 4	SC2
4	35	1	37	2	4 ⟶ 3	SC2
5	5	1	9	3	3 ⟶ 1	SC2
6	23	2	40	1	2 ⟶ 7	SC3
7	39	2	50	1	7 ⟶ 2	SC3

Note. The horizontal speed and vertical speed of stacker are 1.4 s/column and 6 S/layer, respectively.

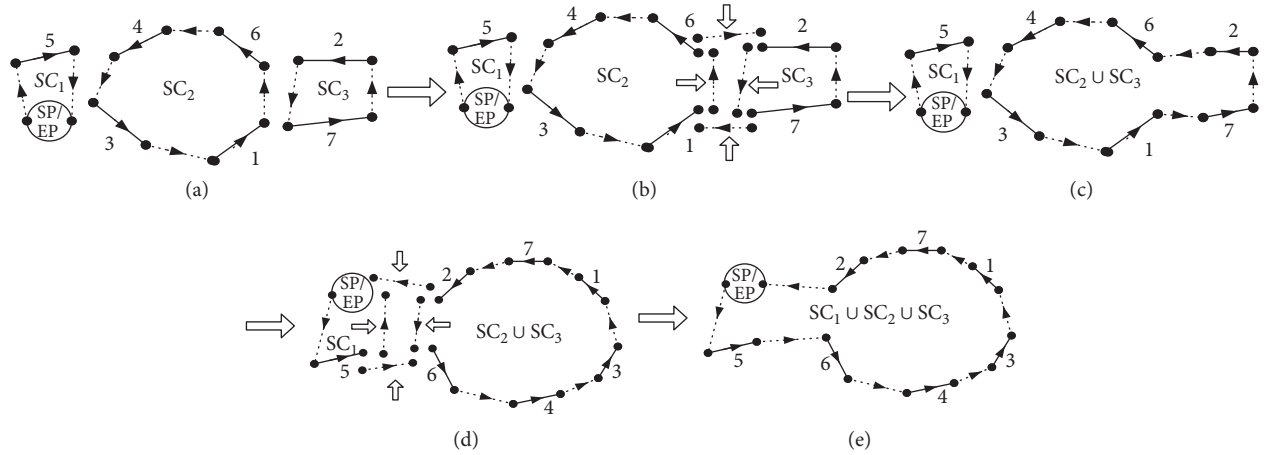

Figure 6: Minimum cost merging process of subloops of the example shown in Table 2. (a) Three subloops determined by assignment. (b) Minimum cost merging scheme. (c) New subloops after minimum cost merging. (d) Minimum cost merging scheme. (e) Tour after minimum cost merging.

The adjacency matrix is created based on the constructed bipartite graph G and the defined no-load travel time. The column index of the matrix corresponds to the serial number of the S/R requests, and the last column is the virtual end point EP. The row vector component of the first line in the matrix represents the running time from the start point of the stacker to the start point of each load travel, and the time arrived to EP is ∞, which is expressed by a relatively large number. The component of the remaining row vector in the matrix represents the running time of the stacker from the end point of the load travel to the start point of other load travels and EP, and the time to reach the end point of the current load travel is ∞, which is represented by a relatively large number. The row index in the matrix is 1 larger than the serial number of the S/R request.

Hungary algorithm is applied to solve the adjacency matrix, and the optimal assignment scheme of bipartite graph shown in Figure 9 is obtained. In bipartite graph, V_1 is composed of the end point of the load travel and the start point of the stacker, and V_2 is composed of the start point of the load travel and the virtual end point of the stacker. Therefore, the edges determined by the optimal assignment scheme are all no-load travel, and the directions are all from the nodes of V_1 to the nodes of V_2. According to the

Table 3: An example of the suboptimal solution of TSP obtained by minimum cost merging.

Number	Load travels				No-load travels determined by assignment	Subloops
	Start point		End point			
	Column	Layer	Column	Layer		
SP/EP	10	1	—	—	SP ⟶ 6	SC1
1	29	4	40	1	6 ⟶ 2	SC1
2	5	1	21	5	2 ⟶ 5	SC1
3	41	4	45	1	5 ⟶ 7	SC1
4	35	1	49	3	7 ⟶ EP	SC1
5	17	2	25	1	1 ⟶ 4	SC2
6	5	1	22	5	4 ⟶ 9	SC2
7	15	1	18	3	9 ⟶ 1	SC2
8	45	3	50	1	3 ⟶ 8	SC3
9	37	5	45	1	8 ⟶ 3	SC3

Note. The horizontal speed and vertical speed of stacker are 1.4 s/column and 6 S/layer respectively.

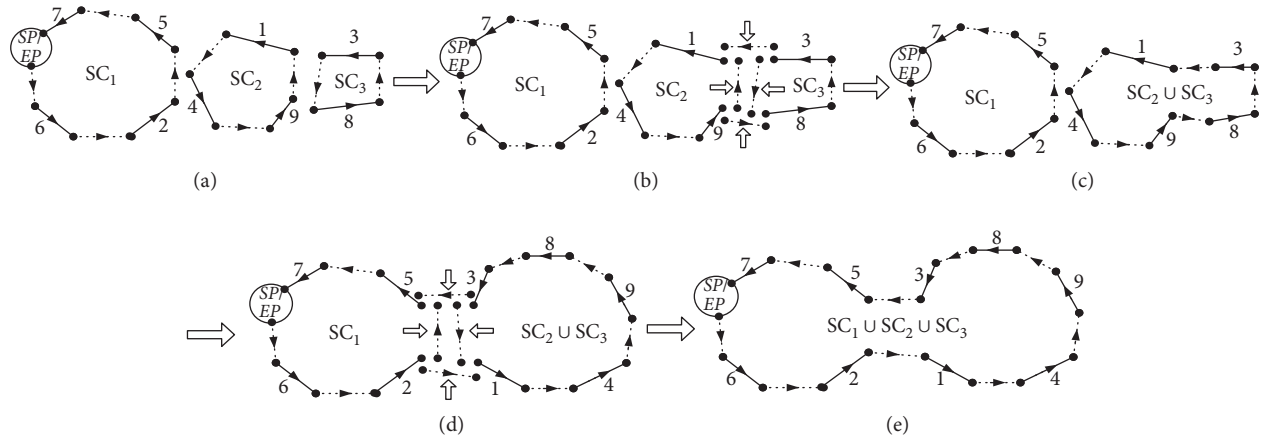

Figure 7: Equivalent merging process of subloops of the example shown in Table 3 (a) Three subloops determined by assignment. (b) Equivalent merging scheme. (c) New subloops after equivalent merging. (d) Minimum cost merging scheme. (e) Tour after minimum cost merging.

corresponding relationship between the serial number of S/R requests and the row and column index in the adjacency matrix, the no-load travels determined by the optimal assignment scheme in Figure 9 are SP ⟶ R_3, S_1 ⟶ R_1, S_2 ⟶ S_4, R_1 ⟶ S_3, S_3 ⟶ R_2, R_2 ⟶ S_2, S_4 ⟶ EP, R_3 ⟶ S_1.

Starting from the start location SP of the stacker, the subsequent S/R requests are determined according to definite no-load travel. At the same time, the definite S/R requests are marked in the S/R requests sequence, and then starting from the definite end location of the S/R requests, the next S/R request is found according to the definite no-load travel and marked until the virtual end of the stacker is found. If the traversal of all the S/R requests nodes is not completed after finding EP, the first unmarked S/R request is found in the S/R requests sequence and starts from the destination corresponding to this S/R request. The above process is repeated until the traversal of all the S/R request nodes is completed. If the no-load travel and S/R requests do not constitute multiple subloop, then starting from SP and finding EP according to the optimal assignment scheme, the traversal of all S/R requests nodes is completed, and the traversal order of all the S/R requests nodes is the optimal solution of the optimization problem of S/R requests sequence. The no-load travel and S/R requests determined by the corresponding optimal assignment scheme of the example shown in Figure 8 do not constitute multiple subloops, and the optimal execution sequence of S/R requests sequence determined by the optimal assignment scheme is shown in Figure 10.

However, when the no-load travel determined by the optimal assignment scheme and the load travel constitute multiple subloops (as shown in Figure 11), series connection of all load travels along determined no-load travel by the stacker cannot form a tour; that is, the sequencing of all S/R requests cannot realize. An equivalent merging algorithm of subloops based on Corollary 2 will be introduced in the next section. The algorithm is embedded in the heuristic algorithm based on assignment. The subloops are merged without changing the total time of the no-load travel, and a tour contacting all load travels is formed.

When the no-load travel determined by the optimal assignment scheme is in series with the load travel, if multiple subloops are formed, then the execution order of the determined S/R requests sequence violates Constraint (4) in the given model. In order to eliminate the subloops

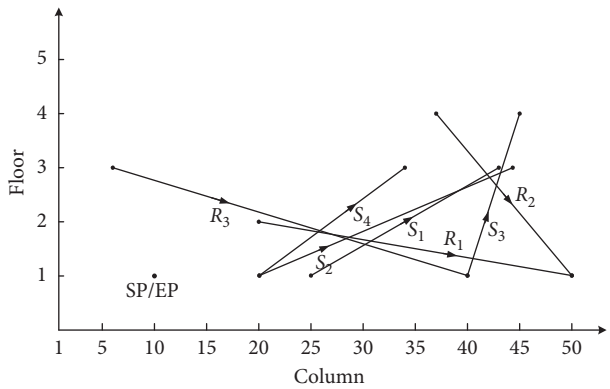

Figure 8: An example of four storage requests and three retrieval requests: the horizontal axis and the vertical axis represent the column and floor, where the stacker access point is located, respectively.

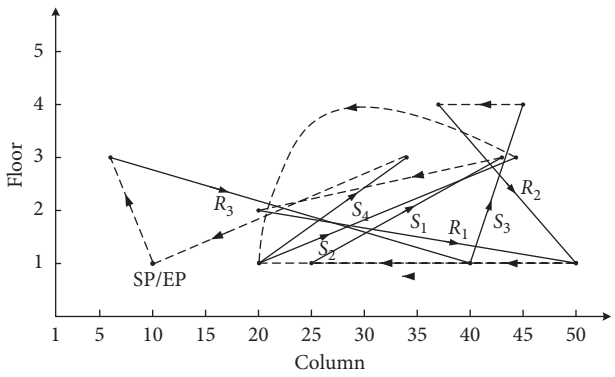

Figure 10: Optimal solution of execution sequence of S/R requests sequence: the horizontal axis and the vertical axis represent the column and floor, where the stacker access point is located, respectively, the solid line represents load travel, and the dotted line represents no-load travel.

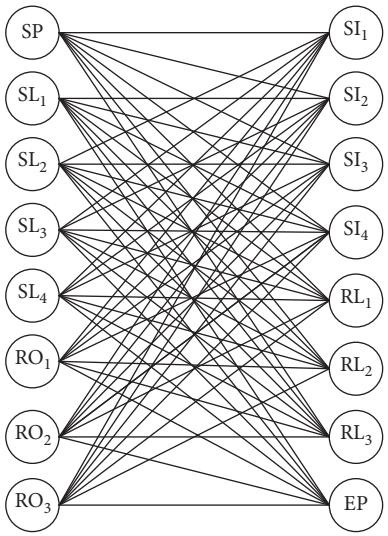

Figure 9: Bipartite graph of four storage requests and three retrieval requests.

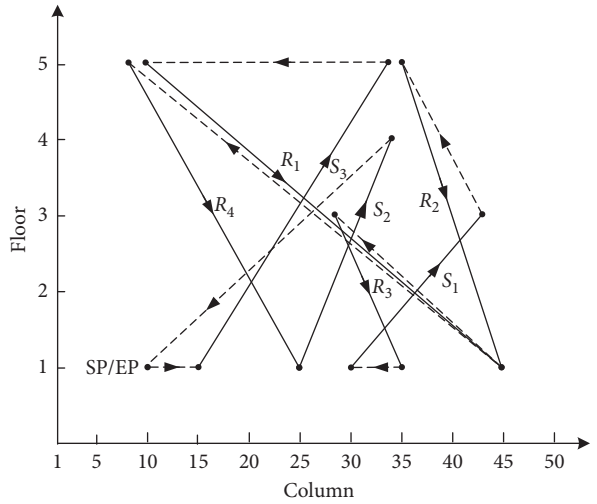

Figure 11: An example with two subloops: the horizontal axis and the vertical axis represent the column and floor, where the stacker access point is located, respectively, the solid line represents load travel, and the dotted line represents no-load travel.

without increasing the operation time of S/R requests sequence, we propose the subloop equivalent merging algorithm to improve the method of concatenating S/R requests nodes. This algorithm is used to consummate the method of connecting S/R request nodes in series. It is connected with the optimal assignment scheme in the assignment based heuristic algorithm and used to solve the following three problems in turn: determining the subloop, constructing the subloop merging matrix, and subloop merging. Firstly, the optimal assignment scheme is used to determine the subloop composed of no-load travel and load travel, so that each load travel (including the start point of the stacker) is connected with the no-load travel determined by the optimal assignment scheme (see Figure 11). There are two subloops in Figure 11, namely, SC_1 and SC_2. The subloop SC_1 is $S_1 \longrightarrow R_2 \longrightarrow R_3 \longrightarrow S_1$ and the subloop SC_2 is $SP \longrightarrow S_3 \longrightarrow R_1 \longrightarrow R_4 \longrightarrow S_2 \longrightarrow SP$. We can implement the sort of S/R requests through the equivalent merging of subloops (Corollary 2 has been proved). The subloop merging matrix is used to record the merging scheme of any two subloops. Figure 12 shows the subloop merging matrix with three subloops. The element in the matrix represents the time change after the merging of the subloops that do not contain the start point of the stacker with any other subloops, and its value is the time of the no-load travel corresponding to the two newly generated edges minus the time of the no-load travel corresponding to the original two edges. Taking Figure 5(b) as an example, the time change is d_{kl} after merging of SC_1 and SC_2, and its value is $t(A,C) + t(D,B) - t(A,B) - t(D,C)$. The element with zero value is called zero element, and the merging scheme corresponding to zero element is a subloop equivalent merging scheme. In the subloop merging matrix, it is forbidden to merge any two no-load travels in the same subloop, which is expressed as ∞. In addition, we define the dashed box in the subloop merging matrix as merging block. Each merging block corresponds to two subloops, and the

number of elements in the merging block represents the number of merging schemes corresponding to the two subloops.

When the subloop merging matrix contains zero elements, the equivalent merging of two subloops corresponding to zero elements can be realized. When the matrix contains multiple zero elements, the merging blocks are taken as units firstly and are sorted according to the number of zero elements. Then, the merging block with the least zero elements is selected, in which all zero elements are picked, and the number of zero elements on the row and column of each zero element is counted and sorted. The two no-load travels corresponding to the zero element with the minimum number in the statistical results are deleted in the merging of subloops. The subsequent merging process of subloops can be found in the proof process of Corollary 2. After a subloop is merged, the elements on the row and column where zero element with the minimum number in the statistical results in the merge block is located are deleted. Repeat the above steps until there are no more subloops. Figure 11 is an example with two subloops. In the merging process of SC_1 and SC_2, the no-load travel $R_2 \longrightarrow R_3$ in SC_1 and the no-load travel $S_2 \longrightarrow SP$ in SC_2 are deleted, and two new no-load travels $S_2 \longrightarrow R_3$ and $R_2 \longrightarrow SP$ are selected, and the tour formed after merging is $SP \longrightarrow S_3 \longrightarrow R_1 \longrightarrow R_4 \longrightarrow S_2 \longrightarrow R_3 \longrightarrow S_1 \longrightarrow R_2 \longrightarrow SP$ (see Figure 13).

Considering that the optimal assignment scheme may construct multiple subloops, subloop equivalent merging algorithm is introduced. By adjusting the no-load travel equivalently, multiple subloops are merged into a time invariant tour. The premise of equivalent merging of subloops is that there are zero elements in the merging matrix of subloops. However, we do not know whether the constructed subloop merging matrix contains zero elements. If the no-load travel determined by the optimal assignment scheme forms multiple subloops when concatenating with the load travel, and there are no zero elements in the subloop merging matrix, that is, there is no equivalent subloop merging scheme (as shown in the example in Figure 14), then it is impossible to get the execution sequence of S/R requests that satisfy Constraint (4) through the subloop equivalent merging algorithm. To solve this problem, we propose the minimum cost heuristic rule to merge the subloops and eliminate multiple subloops without equivalent merging scheme.

In the process of subloops merging, two no-load travels should be deleted, and two new no-load travels should be selected at the same time. Since the two newly selected no-load travels and the existing no-load travels may form an equivalent merging scheme, in order to avoid omitting the equivalent merging scheme, the minimum merging cost heuristic rule requires that only two subloops be merged at a time, and the two subloops with the minimum time increment of no-load travels are required. The minimum merging cost heuristic rule is a supplement to the subloop equivalent merging algorithm, and its general idea is as follows. If there is no zero element in the subloop merging matrix, all the elements in the subloop merging matrix are traversed, and the element with the smallest

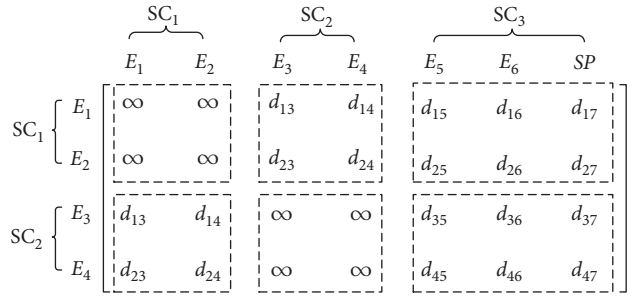

Figure 12: Subloop merging matrix with three subloops.

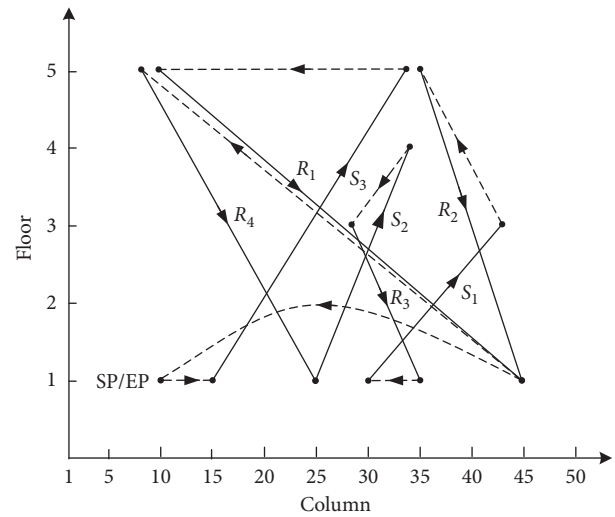

Figure 13: An example of subloops after equivalent merging: the horizontal axis and the vertical axis represent the column and floor, where the stacker access point is located, respectively, the solid line represents load travel, and the dotted line represents no-load travel.

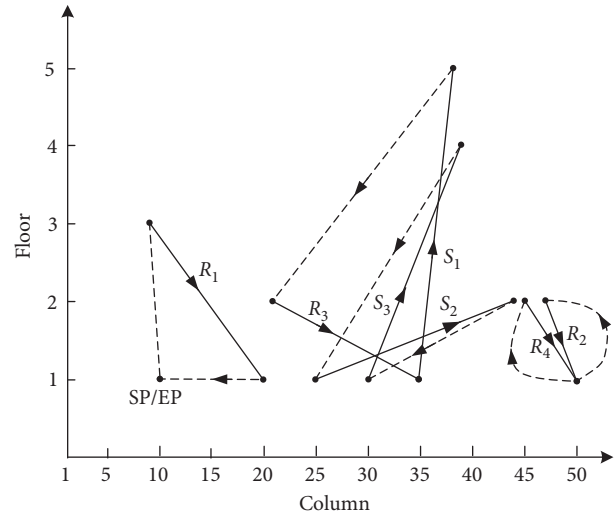

Figure 14: An example of minimum cost merging with four subloops: the horizontal axis and the vertical axis represent the column and floor, where the stacker access point is located, respectively, the solid line represents load travel, and the dotted line represents no-load travel.

time increment is selected. The two subloops corresponding to the merging block, where the element is located, are determined as the two combined subloops, and the two no-load travels corresponding to the element are deleted during the subloop merging process. After the two subloops are merged, the subloop merging matrix will be rebuilt. If there are no zero elements in the matrix, continue with the above steps until there are no more subloops. For example, in the example shown in Figure 14, the optimal assignment scheme determines four subloops, namely, $R_3 \longrightarrow S_1 \longrightarrow R_3$, (SC_1), $R_2 \longrightarrow R_4 \longrightarrow R_2$, (SC_2), $S_2 \longrightarrow S_3 \longrightarrow S_2$, (SC_3), and $SP \longrightarrow R_1 \longrightarrow SP$, (SC_4), and there is no zero element in the subloop merging matrix constructed by these 4 subloops. According to the minimum merging cost heuristic rule, SC_1 and SC_3 are merged first. The deleted no-load travels are $R_3 \longrightarrow S_1$ and $S_2 \longrightarrow S_3$, and the merged subloop is $R_3 \longrightarrow S_3 \longrightarrow S_2 \longrightarrow S_1 \longrightarrow R_3\,(SC_5)$. Then, three subloops, namely, (SC_5), SC_2 and SC_4, construct the subloop merging matrix again, and there is still no zero element in the matrix. SC_5 and SC_2 are merged according to the minimum merging cost heuristic rule. The deleted no-load travels are $S_1 \longrightarrow S_2$ and $R_2 \longrightarrow R_4$, and the merged subloop is $R_3 \longrightarrow S_3 \longrightarrow S_2 \longrightarrow R_4 \longrightarrow R_2 \longrightarrow S_1 \longrightarrow R_3$ (SC_6). There is still no zero element in the subloop merging matrix constructed by SC_6 and SC_4. The last two subloops are merged according to the minimum merging cost heuristic rule. The deleted no-load travels are $R_1 \longrightarrow SP$ and $S_1 \longrightarrow R_3$, and the merged loop is $SP \longrightarrow R_1 \longrightarrow R_3 \longrightarrow S_3 \longrightarrow S_2 \longrightarrow R_4 \longrightarrow R_2 \longrightarrow S_1 \longrightarrow SP$ (see Figure 15).

In summary, after introducing the subloop equivalent merging algorithm and the minimum merging cost heuristic rule, the steps of the assignment based heuristic algorithm are as follows:

Step 0: According to the bipartite graph definition mentioned above, the vertex sets V_1 and V_2 including the start and end points of the stacker and the start and end points of the load travels are created.

Step 1: The adjacency matrix of bipartite graph is constructed according to the no-load travel time defined in Table 1.

Step 2: The Hungarian algorithm is used to solve the optimal assignment scheme of adjacency matrix, and the no-load travel is selected by the optimal assignment scheme.

Step 3: The load travel is connected in series according to the selected no-load travel, and the number of subloops is determined.

Step 4: If there is no subloop, sort the S/R requests according to the optimal assignment scheme, and go to Step 10.

Step 5: The subloop merging matrix shown in Figure 12 is constructed based on the determined subloops.

Step 6: Check whether there are zero elements in the subloop merging matrix.

Step 7: If the subloop merging matrix contains zero elements, the merging blocks are sorted according to the number of zero elements in the merging block, and the subloops are merged equivalently according to the sorting order, and the optimal assignment scheme is updated, and go to Step 9.

Step 8: After traversing all the elements in the subloop merging matrix, the element with the smallest time increment is selected, and the two subloops corresponding to the element are merged according to the minimum merging cost heuristic rule, and the optimal assignment scheme is updated.

Step 9: Count the number of subloops composed of no-load travel and load travel in the updated optimal assignment scheme, and go to Step 4.

Step 10: Calculate the no-load travel time of the obtained S/R requests sequence.

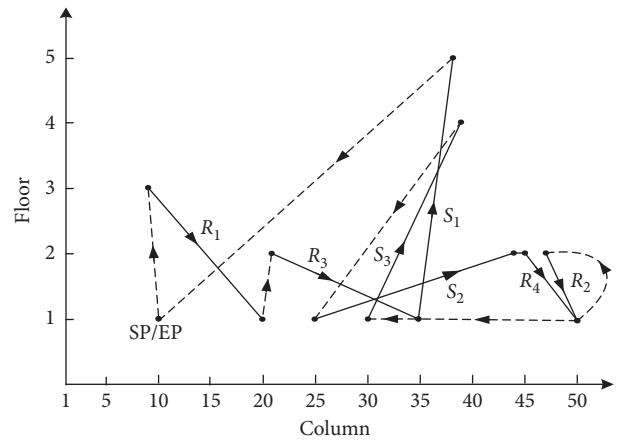

Figure 15: Loops after merging with minimum merging cost heuristic rules: the horizontal axis and the vertical axis represent the column and floor, where the stacker access point is located, respectively, the solid line represents load travel, and the dotted line represents no-load travel.

4. Experimental Results

Table 4 shows the experimental results of assignment based heuristic sorting algorithm and enumeration sorting algorithm. The first column shows the number of S/R requests received by the system. The second column shows the no-load travel running time of the stacker for the best sequence found by the enumeration sorting algorithm. Note that the optimal sequence obtained by the algorithm is the real optimal solution of each test case. The third column shows the computation time of the enumeration sort algorithm (on a 2.5 GHz core (TM) i5 machine). The fourth column shows the no-load travel running time of the stacker of the best sequence found by the proposed assignment based heuristic sorting algorithm. The fifth column shows the computation time of the assignment based heuristic sorting algorithm (on a 2.5 GHz core (TM) i5 machine). The sixth to eighth columns show the number of subloops, the equivalent merging times of subloops, and the minimum cost merging times of subloops generated by the proposed heuristic sorting algorithm based on assignment. The ninth column shows the

time difference of no-load travel running time of the stacker between the optimal sequence generated by the enumeration algorithm and the assignment based heuristic algorithm.

In the 49 test cases, the number of subloops is 0 for 10 cases. For the S/R requests sequence without subloops, the assignment based sorting algorithm connects all the S/R requests in series through the optimal assignment scheme, and the no-load travel time of the sorting sequence is equal to the optimal no-load travel time of the stacker found by enumeration algorithm. When the S/R requests connected in series by assignment schemes form multiple subloops, the assignment based sorting algorithm combines the subloops by equivalent merging and minimum cost merging. Among the 39 cases with nonzero subloops, 14 cases are merged into one tour only by equivalent merging, 7 cases are merged into one tour only by minimum cost merging, and 18 cases are merged into one tour by two methods. Compared with enumeration algorithm, the assignment based sorting algorithm can generate a sequence with the same no-load travel running time of stacker for each case when only equivalent merging method is used to merge subloops.

When the minimum cost merging rule is used in the subloops merging process, the assignment based sorting algorithm generates two suboptimal solutions compared with enumeration algorithm in 25 test cases. In two examples of suboptimal solutions, the merging of subloops is realized by means of equivalent merging and minimum cost merging. Combined with Corollary 1 and Corollary 2, the assignment based sorting algorithm can get the best sequence scheme when only the equivalent merging method is used to merge the subloops, and the number of subloops is zero. When the minimum cost rule is used to merge the subloops, the best sequence scheme can also be obtained in most cases, and the difference of no-load travel time of stacker between the suboptimal solution and the optimal solution is extremely small.

In order to research the effectiveness of the proposed algorithm in large-scale S/R requests sorting problem, we increase the scale of the sequence. For the sequence with more than 15 S/R requests, the time of enumeration sorting algorithm is too long, so the experiment of enumeration sorting is not carried out. Table 5 shows the experimental results of the nearest neighbor heuristic sorting algorithm and the assignment based heuristic sorting algorithm. The number of S/R requests sequences is in the first column. They are two to ten times the scale of the problems in previous experiments. The second column shows the empty running time of the stacker for the best sequence found by the nearest neighbor heuristic sorting algorithm. The third column shows the computation time of the nearest neighbor algorithm (on a 2.5 GHz core (TM) i5 machine). The ninth column and the tenth column are the performance comparison between the nearest neighbor heuristic sorting algorithm and the algorithm proposed in this paper.

Among the 49 test cases, 45 cases obtained the best sequence of large-scale S/R requests by the optimal assignment scheme and equivalent merging subloops, and the remaining 4 cases were the sequences of large-scale S/R requests obtained by equivalent merging and minimum cost merging. With the increase of the scale of the S/R requests sequence, the probability of the subloops equivalent merging increases. Only a few cases do not have the subloops equivalent merging scheme. This is because, with the increase of the scale of the S/R requests sequence, the no-load travel in the subloop also increases, and the equivalent merging schemes in the constructed adjacency matrix increase correspondingly. Therefore, most of the large-scale S/R requests sequence cases can merge the subloops by equivalent merging to find the optimal sorting sequence of the S/R requests.

Compared with the nearest neighbor heuristic sorting algorithm, the sorting algorithm based on assignment can generate better sorting sequence, which can save up to 34.38% of the no-load running time. When the S/R requests scale is 30 to 50, the no-load travel time can be saved by 3.23% on average. With the increase of the scale of the S/R requests, the calculation time of the assignment based sorting algorithm does not increase significantly. In the near-real-time (less than 0.4 seconds), a better sorting sequence for each test case is found.

In order to make readers have a deeper understanding of the current status of related research fields, we compare the optimization method proposed in this paper with the existing optimization methods. Due to the differences of different optimization methods in storage system layout, stacker performance, and command cycle mode, it is difficult to fully reflect the advantages and disadvantages of various algorithms only through the calculation time of algorithm and the saved operation time comparing with the nearest neighbor algorithm. However, the comparison of the above performance indicators can partially or indirectly reflect the advantages and disadvantages of different algorithms. Based on this consideration, the performance comparison between the existing sorting algorithms for S/R requests and our algorithm is listed in Table 6. In the scenarios of different number of I/O depot, different command cycle modes, and different scale of S/R request sequence, the completion time and the calculation time of S/R requests are compared between different algorithms and the nearest neighbor algorithm.

It can be seen from Table 6 that various optimization algorithms can improve the completion time of S/R requests obtained by the nearest neighbor algorithm to varying degrees. This is because the nearest neighbor algorithm is a local optimal algorithm, while the other algorithms are global optimization algorithms, which can optimize the execution order of S/R requests from a global perspective and reduce the no-load travel time of the stacker. In addition, the heuristic algorithm has a good performance in computing time.

Table 4: Comparisons of unload times between EA and proposed method for small-scale problem.

N_r	Enumeration		Assignment					Diff[1]
	T_NL[a]	CPU	T_NL[b]	CPU	N_SL	EMT	MinMT	
7	177.8	0.016	177.8	0.006	2	0	1	0
7	188.2	0.004	188.2	0.003	0	0	0	0
7	181.8	0.015	181.8	0.003	3	1	1	0
7	244.8	0.016	244.8	0.004	2	1	0	0
7	208.2	0.015	208.2	0.004	0	0	0	0
7	193.2	0.016	193.2	0.004	3	0	2	0
7	168.6	0.016	168.6	0.003	3	1	1	0
7	249.6	0.015	249.6	0.003	2	0	1	0
7	167.4	0.016	167.4	0.004	3	0	2	0
7	203.0	0.015	203.0	0.002	2	1	0	0
8	213.0	0.043	213.0	0.004	2	0	1	0
8	219.8	0.045	219.8	0.004	0	0	0	0
8	301.4	0.054	301.4	0.004	2	1	0	0
8	210.4	0.035	210.4	0.007	4	1	2	0
8	238.4	0.047	238.4	0.002	3	1	1	0
8	220.8	0.062	220.8	0.016	0	0	0	0
8	246.4	0.046	246.4	0.016	3	2	0	0
8	275.8	0.047	275.8	0.002	0	0	0	0
9	243.2	0.330	243.2	0.008	3	1	1	0
9	230.6	0.323	230.6	0.004	4	2	1	0
9	255.6	0.323	255.6	0.002	2	0	1	0
9	285.8	0.329	285.8	0.004	0	0	0	0
9	212.0	0.312	212.0	0.006	3	2	0	0
9	253.4	0.329	253.4	0.003	3	1	1	0
9	240.4	0.342	240.4	0.002	0	0	0	0
9	201.8	0.338	201.8	0.003	2	1	0	0
9	266.0	0.316	266.0	0.003	0	0	0	0
9	247.8	0.311	247.8	0.005	3	2	0	0
9	239.8	0.326	241.2	0.004	3	1	1	1.4
10	229.8	3.385	229.8	0.016	3	2	0	0
10	292.8	3.229	292.8	0.003	0	0	0	0
10	232.4	3.245	232.4	0.016	3	1	1	0
10	349.0	3.198	349.0	0.002	2	1	0	0
10	271.0	3.213	271.0	0.002	3	1	1	0
10	332.4	3.229	332.4	0.002	2	1	0	0
10	291.6	3.230	291.6	0.015	3	2	0	0
10	272.6	3.229	272.6	0.002	2	1	0	0
10	280.8	3.182	280.8	0.015	0	0	0	0
10	241.8	3.244	241.8	0.012	3	0	2	0
10	251.6	3.354	251.6	0.015	3	1	1	0
11	317.4	49.134	317.4	0.016	4	2	1	0
11	243.4	51.510	243.4	0.016	3	1	1	0
11	286.2	45.344	286.2	0.015	2	1	0	0
11	267.8	41.389	267.8	0.016	3	1	1	0
11	283.2	48.496	283.2	0.015	3	1	1	0
11	267.8	42.226	267.8	0.016	3	1	1	0
11	283.2	49.931	283.2	0.015	3	1	1	0
11	294.8	40.909	294.8	0.015	2	1	0	0
11	300.4	44.486	302.0	0.015	3	1	1	1.6

Note. Five-layer double row shelf with 15 I/O depots, N_r: number of S/R requests; T_NL: no-load travel time of stacker (sec); CPU: computation time (sec); N_SL: the number of subloops; EMT: the equivalent merging times of subloops; MinMT: the minimum cost merging times of subloops; $Diff^{1} = T\_NL^{a} - T\_NL^{b}$.

Table 5: Comparisons of unload times between NN and proposed method for large-scale problem.

N_r	Nearest neighbor		Assignment					Diff[1]	Diff[2] (%)
	T_NL[a]	CPU	T_NL[b]	CPU	N_SL	EMT	MinMT		
15	458.2	0.0224	429.2	0.016	15	458.2	0.0224	29	6.33
15	382.0	0.0284	366.6	0.016	3	2	0	15.4	4.20
15	355.2	0.0262	339.4	0.062	3	1	1	15.8	4.66
15	513.6	0.0248	444.6	0.016	2	1	0	69	15.52
15	513.6	0.0268	382.2	0.016	2	1	0	131.4	34.38
15	451.8	0.0287	416.2	0.016	0	0	0	35.6	8.55
15	519.2	0.0291	480.4	0.015	2	1	0	38.8	8.08
15	487.8	0.0285	469.2	0.016	3	2	0	18.6	3.96
15	401.4	0.0279	386.2	0.015	3	1	1	15.2	3.94
15	550.6	0.0261	521.2	0.016	0	0	0	29.4	5.64
20	613.6	0.0268	561.4	0.016	3	2	0	52.2	9.30
20	612.6	0.0259	581.4	0.016	2	1	0	31.2	5.37
20	612.6	0.0268	569.4	0.016	2	1	0	43.2	7.59
20	726.8	0.0261	709.4	0.015	0	0	0	17.4	2.45
20	573.6	0.0223	551.2	0.016	3	2	0	22.4	4.06
20	599.6	0.0272	585.2	0.016	2	1	0	14.4	2.46
20	607.4	0.0249	576.4	0.015	0	0	0	31	5.38
20	636.2	0.0259	602.4	0.016	3	1	1	33.8	5.61
30	884.8	0.0351	865.4	0.016	2	1	0	19.4	2.24
30	1007.6	0.0335	944.0	0.031	2	1	0	63.6	6.74
30	950.8	0.0373	928.4	0.047	2	1	0	22.4	2.41
30	850.4	0.0395	842.8	0.031	3	2	0	7.6	0.90
30	810.8	0.0362	785.2	0.031	3	2	0	25.6	3.26
30	878.6	0.0332	840.6	0.031	2	1	0	38	4.52
30	911.0	0.0387	862.6	0.015	2	1	0	48.4	5.61
30	859.6	0.0462	832.6	0.031	2	1	0	27	3.24
50	1536.6	0.0442	1499.4	0.078	3	2	0	37.2	2.48
50	1610.8	0.0418	1561.8	0.094	3	2	0	49	3.14
50	1469.4	0.0424	1418.6	0.047	0	0	0	50.8	3.58
50	1470.4	0.0455	1447.0	0.063	2	1	0	23.4	1.62
50	1395.0	0.0448	1344.0	0.047	2	1	0	51	3.79
50	1373.8	0.0472	1322.0	0.063	2	1	0	51.8	3.92
50	1563.8	0.0475	1497.2	0.062	2	1	0	66.6	4.45
50	1695.2	0.0456	1670.2	0.078	2	1	0	25	1.50
50	1461.8	0.0429	1447.0	0.047	2	1	0	14.8	1.02
50	1268.0	0.0505	1221.6	0.047	2	1	0	46.4	3.80
80	2185.6	0.0611	2118.2	0.156	3	2	0	67.4	3.18
80	2386.8	0.0668	2359.8	0.171	3	1	1	27	1.14
80	2236.4	0.0634	2196.0	0.14	2	1	0	40.4	1.84
80	2281.6	0.0602	2244.6	0.125	0	0	0	37	1.65
80	2235.6	0.0636	2190.4	0.156	2	1	0	45.2	2.06
80	2367.2	0.0629	2310.4	0.156	2	1	0	56.8	2.46
80	2311.2	0.0695	2247.8	0.187	3	2	0	63.4	2.82
80	2389.2	0.0615	2359.8	0.187	3	2	0	29.4	1.25
80	2450.6	0.0658	2405.4	0.203	2	1	0	45.2	1.88
100	2949.6	0.0748	2891.6	0.328	3	2	0	58	2.01
100	3169.4	0.0758	3069.8	0.312	2	1	0	99.6	3.24
100	3081.0	0.0746	3028.8	0.234	2	1	0	52.2	1.72
100	2845.8	0.0837	2795.0	0.359	3	2	0	50.8	1.82

Note. Five-layer double row shelf with 15 I/O depots; N_r: number of S/R requests; T_NL: no-load travel time of stacker (sec); CPU: computation time (sec); N_SL: the number of subloops; EMT: the equivalent merging times of subloops; MinMT: the minimum cost merging times of subloops; $\text{Diff}^1 = \text{T\_NL}^a - \text{T\_NL}^b$; $\text{Diff}^2 = (\text{T\_NL}^a - \text{T\_NL}^b)/\text{T\_NL}^a$.

Table 6: Comparisons of existing methods with the proposed method.

Studies	N_r	N_d	CR	T_NN	Contrastive algorithm			GAP
					Algorithm	T _A	CPU	
Gharehgozli et al. [27]	20	10	2	1113.10	TPHA	976.42	0.0177	136.68 s
Gharehgozli et al. [27]	50	10	2	2730.60	TPHA	2325.7	0.0397	404.90 s
Gharehgozli et al. [27]	100	10	2	5406.32	TPHA	4650.2	0.1321	756.12 s
Gharehgozli et al. [27]	150	10	2	8006.24	TPHA	6965.1	0.2973	1041.14 s
Gharehgozli et al. [27]	200	10	2	10692.10	TPHA	9190.7	0.5462	1501.4 s
Gharehgozli et al. [27]	100	10	1	6312.60	TPHA	6130.3	0.1147	182.30 s
Gharehgozli et al. [27]	100	10	1	6219.10	TPHA	6197.7	0.0942	21.40 s
Van den Berg and Gademann [28]	10	1	2	7.80	PA	7.3	—	0.5 UT
Van den Berg and Gademann [28]	20	1	2	14.70	PA	14.0	—	0.7 UT
Van den Berg and Gademann [28]	40	1	2	28.30	PA	27.1	—	1.2 UT
Van den Berg and Gademann [28]	10	2	1or2	11.30	PA	11.1	—	0.2 UT
Van den Berg and Gademann [28]	20	2	1or2	22.60	PA	22.4	—	0.2 UT
Van den Berg and Gademann [28]	40	2	1or2	45.40	PA	45.0	—	0.4 UT
Gharehgozli et al. [27]	10	2	2	544.69	PA	449.06	0.61	95.63 s
Gharehgozli et al. [27]	20	2	2	1048.19	PA	820.91	20.27	227.28 s
Gharehgozli et al. [27]	30	2	2	1609.64	PA	1240.39	187.04	369.25 s
Gharehgozli et al. [27]	40	2	2	2146.66	PA	1633.73	776.24	512.93 s
Gharehgozli et al. [27]	50	2	2	2603.50	PA	2005.75	1811.57	597.75 s
Popovic et al. [21]	12	1	6	343.60	ISL	312.84	—	30.76 s
Popovic et al. [21]	12	1	6	343.60	BH	307.81	<1	35.79 s
Popovic et al. [21]	12	1	6	343.60	GA	282.21	<20	61.39 s
Popovic et al. [21]	24	1	6	660.87	ISL	593.20	—	67.67 s
Popovic et al. [21]	24	1	6	660.87	BH	588.66	<16	72.21 s
Popovic et al. [21]	24	1	6	660.87	GA	521.09	<60	139.78 s
This paper	15	15	—	800.78	HA	760.96	0.0204	39.82 s
This paper	20	15	—	1088.40	HA	1057.70	0.0158	30.70 s
This paper	30	15	—	1588.88	HA	1557.38	0.0291	31.50 s
This paper	50	15	—	2647.08	HA	2604.80	0.0626	42.28 s
This paper	80	15	—	4134.93	HA	4089.18	0.1646	45.76 s
This paper	100	15	—	5360.85	HA	5295.70	0.3083	65.15 s

Note. N_r: number of S/R requests; N_d: number of I/O depots; CR: number of command cycle; T_NN: completion time obtained by nearest neighbor algorithm (sec); T_A: completion time obtained by specified algorithm (sec); CPU: computation time (sec); GAP: T_NN-T_A (sec); TPHA: Two-phase heuristic algorithm; PA: polynomial algorithm; ISL: improved shortest leg heuristic; BH: best heuristic; GA: genetic algorithm; HA: heuristic algorithm based on assignment.

5. Conclusions

In this paper, we address the scheduling problem of large-scale S/R requests sequence in multi-I/O depots AS/RS, in which the load travel time of stacker is fixed, the goods can enter and exit the system through multi-I/O depots, and the stacker can load at most one cargo at any time. Each load travel is considered as an S/R requests vertex, and a special traveling salesman problem is constructed. By introducing the concept of assignment, the adjacency matrix for subloops merging is established, and the sorting method based on the optimal assignment scheme, the equivalent merging method of subloops, and the minimum cost merging method of subloops are proposed.

Experimental results indicate their efficiency. Due to the particularity of the TSP, the sorting method based on the optimal assignment scheme can generate the best solution of the test case without subloops. The proposed equivalent merging method of subloops realizes the merging of subloops with equivalent merging scheme without increasing the no-load travel time. In the process of subloops merging, in order to merge the subloops that cannot be merged by equivalent merging method, the minimum cost merging method is also considered, and the proposed algorithm is modified according to the new subloops merging rules.

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.

Acknowledgments

This work was supported by National Key R&D Program of China (No. 2018YFB1201602) and the National Natural Science Foundation of China under Grant No. 61563029.

References

[1] B. Rouwenhorst, B. Reuter, V. Stockrahm, G. J. van, R. Mantel, and W. H. M. Zijm, "Warehouse design and control: framework and literature review," *European Journal of Operational Research*, vol. 122, no. 3, pp. 515–533, 2000.

[2] J. P. Van den Berg, "A literature survey on planning and control of warehousing systems," *IIE Transactions*, vol. 31, no. 8, pp. 751–762, 1999.

[3] R. D. Meller and A. Mungwattana, "Multi-shuttle automated storage/retrieval systems," *IIE Transactions*, vol. 29, no. 10, pp. 925–938, 1997.

[4] 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.

[5] L. Lin, S. W. Shinn, M. Gen, and H. Hwang, "Network model and effective evolutionary approach for AGV dispatching in manufacturing system," *Journal of Intelligent Manufacturing*, vol. 17, no. 4, pp. 465–477, 2006.

[6] H. N. Bessenouci, Z. Sari, and L. Ghomri, "Metaheuristic based control of a flow rack automated storage retrieval system," *Journal of Intelligent Manufacturing*, vol. 23, no. 4, pp. 1157–1166, 2012.

[7] J. Huh, M.-j. Chae, J. Park, and K. Kim, "A case-based reasoning approach to fast optimization of travel routes for large-scale AS/RSs," *Journal of Intelligent Manufacturing*, vol. 30, no. 4, pp. 1765–1778, 2017.

[8] W. H. Hausman, L. B. Schwarz, and S. C. Graves, "Optimal storage assignment in automatic warehousing systems," *Management Science*, vol. 22, no. 6, pp. 629–638, 1976.

[9] S. C. Graves, W. H. Hausman, and L. B. Schwarz, "Storage-retrieval interleaving in automatic warehousing systems," *Management Science*, vol. 23, no. 9, pp. 935–945, 1977.

[10] J. Blazewicz, K. H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz, "Handbook on scheduling: from theory to applications," *Journal of Scheduling*, vol. 12, no. 4, pp. 433-434, 2009.

[11] J. Du and J. Y.-T. Leung, "Minimizing total tardiness on one machine is NP-hard," *Mathematics of Operations Research*, vol. 15, no. 3, pp. 483–495, 1990.

[12] H. F. Lee and S. K. Schaefer, "Sequencing methods for automated storage and retrieval systems with dedicated storage," *Computers & Industrial Engineering*, vol. 32, no. 2, pp. 351–362, 1997.

[13] Y. Yu and R. B. M. De Koster, "Sequencing heuristics for storing and retrieving unit loads in 3D compact automated warehousing systems," *IIE Transactions*, vol. 44, no. 2, pp. 69–87, 2012.

[14] M. Eben-Chaime and N. Pliskin, "An integrative model for automatic warehousing systems," *International Journal of Computer Integrated Manufacturing*, vol. 9, no. 4, pp. 286–292, 1996.

[15] M. Eben-Chaime and N. Pliskin, "Operations management of multiple machine automatic warehousing systems," *International Journal of Production Economics*, vol. 51, no. 1-2, pp. 83–98, 1997.

[16] A. Keserla and B. A. Peters, "Analysis of dual-shuttle automated storage/retrieval systems," *Journal of Manufacturing Systems*, vol. 13, no. 6, pp. 424–434, 1994.

[17] B. R. Sarker, A. Sabapathy, A. M. Lal, and M.-H. Han, "Performance evaluation of a double shuttle automated storage and retrieval system," *Production Planning & Control*, vol. 2, no. 3, pp. 207–213, 1991.

[18] B. R. Sarker, L. Mann, and J. R. G. Leal Dos Santos, "Evaluation of a class-based storage scheduling technique applied to dual-shuttle automated storage and retrieval systems," *Production Planning & Control*, vol. 5, no. 5, pp. 442–449, 1994.

[19] S. Tanaka and M. Araki, "An exact algorithm for the input/output scheduling problem in an end-of-aisle multi-shuttle automated storage/retrieval system with dedicated storage," *Transactions of the Society of Instrument and Control Engineers*, vol. 42, no. 9, pp. 1058–1066, 2006.

[20] S. Tanaka, "A hybrid algorithm for the input/output scheduling problem of multi-shuttle AS/RSs," in *Proceedings of International Conference on Instrumentation, Control and Information Technology*, Kagawa, Japan, October 2007.

[21] D. Popovic, M. Vidovic, and N. Bjelic, "Application of genetic algorithms for sequencing of AS/RS with a triple-shuttle module in class-based storage," *Flexible Services and Manufacturing Journal*, vol. 26, pp. 432–453, 2014.

[22] S. Tanaka and M. Araki, "Routing problem under the shared storage policy for unit-load automated storage and retrieval systems with separate input and output points," *International Journal of Production Research*, vol. 47, no. 9, pp. 2391–2408, 2009.

[23] P. Yang, L. Miao, Z. Xue, and B. Ye, "Variable neighborhood search heuristic for storage location assignment and storage/retrieval scheduling under shared storage in multi-shuttle automated storage/retrieval systems," *Transportation Research Part E: Logistics and Transportation Review*, vol. 79, pp. 164–177, 2015.

[24] L. M. Pohl, R. D. Meller, and K. R. Gue, "An analysis of dual-command operations in common warehouse designs," *Transportation Research Part E: Logistics and Transportation Review*, vol. 45, no. 3, pp. 367–379, 2009.

[25] M. Schleyer and K. Gue, "Throughput time distribution analysis for a one-block warehouse," *Transportation Research Part E: Logistics and Transportation Review*, vol. 48, no. 3, pp. 652–666, 2012.

[26] K. R. Gue and G. Meller, "A unit-load warehouse with multiple pickup and deposit points and non-traditional aisles," *Transportation Research Part E: Logistics and Transportation Review*, vol. 48, no. 4, pp. 795–806, 2012.

[27] A. H. Gharehgozli, Y. Yu, X. Zhang, and R. d. Koster, "Polynomial time algorithms to minimize total travel time in a two-depot automated storage/retrieval system," *Transportation Science*, vol. 51, no. 1, pp. 19–33, 2017.

[28] J. P. Van den Berg and A. J. R. M. Gademann, "Optimal routing in an automated storage/retrieval system with dedicated storage," *IIE Transactions*, vol. 31, no. 5, pp. 407–415, 1999.

[29] N. Boysen and K. Stephan, "A survey on single crane scheduling in automated storage/retrieval systems," *European Journal of Operational Research*, vol. 254, no. 3, pp. 691–704, 2016.