ICIC Express Letters
Volume **15**, Number **9**, September **2021**

ICIC International ©2021 ISSN 1881-803X
pp. **991–997**

AN ORDER PICKING ALGORITHM FOR VERTICALLY STACKED AND TOP-RETRIEVAL STORAGE SYSTEMS

Changmuk Kang
Department of Industrial and Information Systems Engineering
Soongsil University
369 Sangdo-ro, Dongjak-gu, Seoul 06978, Korea
ckang@soongsil.ac.kr

Received January 2021; accepted March 2021

Abstract. *Automated and robotic warehouses are becoming more common in the logistics industry. This study considers a variant of autonomous vehicle based storage and retrieval (AVS/R) system in which a transfer robot stores and retrieves vertically stacked bins on top of a grid of stacks. This system maximizes space utilization while spending waste time for removing upper bins if a retrived bin is stored deeply down in a stack. Such a process is called reshuffling. This study proposes an algorithm to determine a sequence of picking orders that minimizes reshuffling. Instead of finding the optimal sequence among exponentially many sequences, this study proposes an efficient greedy algorithm that achieves a heuristic solution. In the numerical study, the best solution was 12.7σ better than the random sequence average; it outperforms most arbitrary sequences.*
Keywords: Commonality, Variety, Constraint, Supplier involvement

1. **Introduction.** Automated and robotic warehouses are becoming more common in the logistics industry [1]. Where conventional systems use aisle-captive cranes, recent systems use multi-deep racks to enhance storage density, in other words, space utilization. According to de Koster [1], recent multi-deep systems can be classified into autonomous vehicle-based storage and retrieval (AVS/R), shuttle-based storage and retrieval (SBS/R), and robotic moveable rack (RMR) systems. This study considers a variant of AVS/R system, which uses "autonomous unit load robots that move on a grid of frame above the storage stacks [1]". This system is implemented by AutoStore$^{\text{TM}}$, which is illustrated in Figure 1. This system maximizes space utilization by vertically stacking storage bins without aisles and retrieving them from the top. A disadvantage of this system is wasting retrieval time due to *reshuffling*. All the upper (blocking) bins should be retrieved first before retrieving a designated bin. Their depth levels are reshuffled during this process. Since reshuffling time takes significant portion of the entire retrieval process, minimizing the number of reshuffling moves greatly contributes to enhancing efficiency of this system. This study proposes an algorithm to determine a sequence of picking orders to minimize it. A process of picking up an order is a series of bin retrieval operations.

In general AVS/R and SBS/R systems, the system performance metrics, which are mainly throughput time and space utilization, have been evaluated by queueing network models [2,3]. They are, however, single-lane or dual-deep systems in which each bin can be directly accessed by a vehicle or a shuttle. Whereas some studies [4,5] consider multi-deep systems, those systems do not require reshuffling because all items are assumed identical. Almassri et al. [6] evaluate costs of pickup routes in such systems with respect to item storage configuraions; zone-specific and hit-rate based configurations.

Specifically targeted for the AutoStore$^{\text{TM}}$ system, Zou et al. [7] analytically study efficiency of storage policies considering reshuffling. They compare the system performance

DOI: 10.24507/icicel.15.09.991

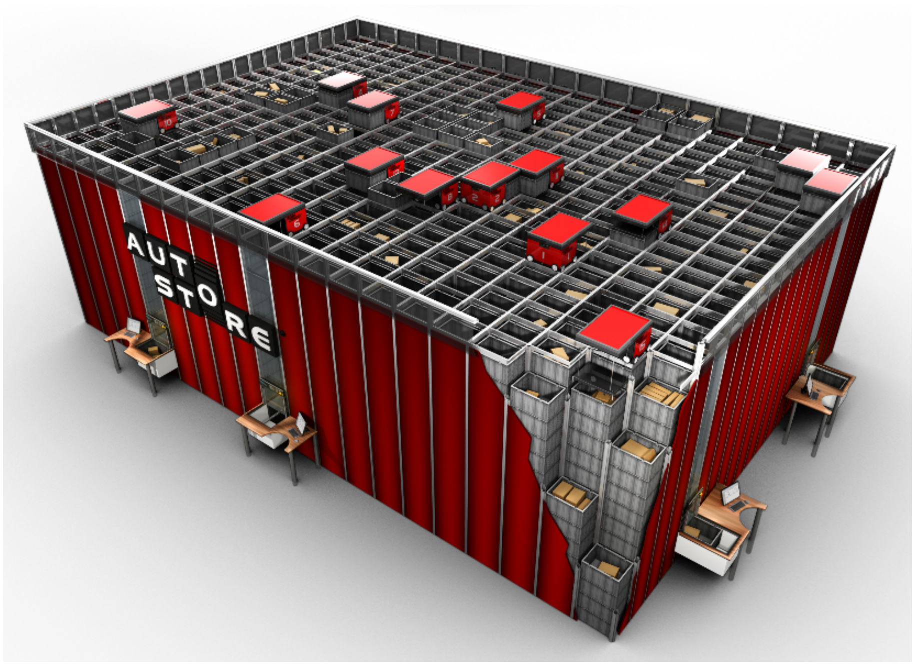

FIGURE 1. Schematic view of the AutoStore^{TM} system (image source: itersnews.com)

in perspectives of dedicated versus shared storage policies, random versus zoned storage policies, and immediate and delayed reshuffling policies. They modeled the system with a semi-open queueing network (SOQN) and estimated expected system throughput under these policies. Because this model compares the policies at a system-design level, it only considers the expected time for reshuffling rather than computing the minimum time for given orders. Tjeerdsma [8] proposes a redesign of the system's order packing line, which is independent of order picking operations.

Reshuffling has been widely studied in container yards rather than warehouses. It takes significant time during retrieving a target container from many vertically stacked containers. The literature has studied relocating blocking containers to other stacks to minimize reshuffling [9,10], and determining loading locations anticipating future retrieval [11-14]. Those problems are modeled as mixed integer programs and solved by heuristic algorithms. A comprehensive review can be found in Lehnfeld and Knust [15].

A warehouse version of the reshuffling problem is easier on the one hand but more complicated on the other hand. The storage bins, which are much lighter than containers, have buffer space on the top floor of the storage stacks. The blocking bins temporarily stay on the top roof floor and come back to the original stack after a target bin is retrieved; no relocation is needed. On the other hand, our problem requires order-based picking that is not considered in the container yard. Especially for warehouses to service online shopping orders, an order consists of multiple stock keeping units (SKU), whereas containers are retrieved one by one. If an order is picked up right now, all its constituting SKUs, which are stored in bins at various locations, have to be retrieved once. The proposed algorithm derives an optimal sequence of processing orders taking this constraint into account.

This study is organized as follows. Section 2 describes detailed operations of the AVS/R system and formally defines the problem. Section 3 presents the fundamental principles of the optimal solutions, and Section 4 develops an algorithm. The algorithm performance is numerically demonstrated in Section 5, and Section 6 concludes the study.

2. **Problem Definition.** A warehouse is an $(l \times m)$ grid of stacks, each of which has maximum n storage bins. A bin is denoted by j, and $\max(j) < l \times m \times n$. One bin stores a single SKU. A bin is stacked from the bottom supported by below bins or the ground. Every stack may have different heights. A stack of bin j is denoted by $T(j)$, and its depth is denoted by $d(j)$ where $d(j)$ is the number of bins stacked above it; $d(j) = 0$ if it is on top of the stack, and $d(j) = n - 1$ if it is at the bottom and the stack is fully charged. A bin can stay at the roof floor without bottom support, but only temporarily to lend access to lower bins.

A horde of transfer robots moves upon the roof floor of the grid. They load and unload bins from the top with a hoist to lift them. A robot can transfer only one bin at a time. There are multiple ports to store and retrieve items. For storage, an item is packed into a bin, and a transfer robot picks the bin at a port and loads it to a designated stack. Then it is stored at the highest (lowest depth) position. Retrieval is a reverse operation. A bin at the top is picked up from a stack and delivered to a port.

Reshuffling occurs if a target bin is below other bins. Those blocking bins are first picked up and stay on the roof floor of near stacks until the target bin is retrieved. They come back to the original stack in the same order of retrieval, and after item pickup at the port, the target bin comes back on top of them; if bin D is retrieved from a stack (A, B, C, D, E), the stack is reshuffled to (D, A, B, C, E) after finishing the retrieval. This case counts the number of reshuffling moves as six by two moves (retrieval and return) each for blocking bins A, B, C.

This study assumes a warehouse from which a set of orders $R = \{r_1, \ldots, r_q\}$ is waiting to be picked up. The problem is to find the optimal sequence to serve all orders with minimum reshuffling moves. An order is composed of different SKU items. Each item is stored in a certain bin, and the required quantity is less than the quantity that the bin stores. Then, without loss of generality, an order r_i is represented as an unordered set of distinct bins, such as $r_i = \{j | \text{order } i \text{ demands bin } j\}$. Although bins within an order can be retrieved in any sequence, retrieval for another order cannot start before finishing packing up this order. If the same bin is required by two different orders, it must be retrieved twice for each order. Where order r_i demands k_i bins, total $K = \Sigma_{i=1,\ldots,q} k_i$ retrievals are required without reshuffling. It is also the size of a solution vector. The number of feasible solutions is $q! \times \Pi_{i=1,\ldots,q} k_i!$, which is smaller than $K!$, but still huge.

3. **Fundamental Principles.** This study represents a solution vector of size K with an order sequence s and within-order bin sequences $t(i)$'s. A complete solution vector is obtained by concatenating $t(i)$'s in an order of s. There are two principles to construct an optimal solution.

Principle 3.1. *For two bins j_1 and j_2 where $d(j_1) < d(j_2)$, they should be retrieved in an order of (j_1, j_2).*

If they are in different stacks, $T(j_1) \neq T(j_2)$, the number of reshuffling is independent of their retrieval sequence. If they are in the same stack, $T(j_1) = T(j_2)$, the number reshuffling moves to retrieve bin j_1 (or j_2) is $2d(j_1)$ (or $2d(j_2)$). If they are retrieved in a sequence of (j_1, j_2), $2d(j_1) + 2d(j_2)$ moves are required. Otherwise, if j_2 comes first, depth of j_1 becomes $d(j_1) + 1$ because the retrieved bin comes back on top of the stack. The total reshuffling moves are $2d(j_2) + 2d(j_1) + 2$. Bin j_1 should be retrieved before j_2 to minimize reshuffling moves. This relationship holds for every pair of bins in the same stack. From Principle 3.1, an optimal within-order sequence is retrieving bin j in ascending order of $d(j)$.

The principle also holds between orders. If possible, it is optimal to retrieve an order with upper bins first and an order with lower bins later. As shown in Figure 2, however, it is often impossible. The bins are noted by capital letters and the numbers in the

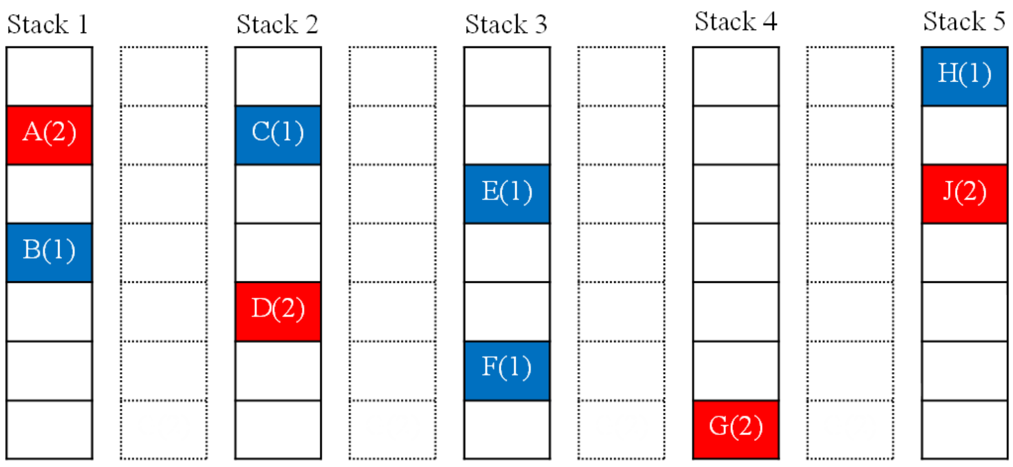

FIGURE 2. Conflicting order sequences

parenthesis denote orders. For orders 1 and 2, target bins are in five stacks and six of them share three stacks. Their within-order sequences are obtained by Principle 3.1 as [H, C, E, B, F] and [A, J, D, G], respectively. If order 1 is first picked up, the whole sequence is [H, C, E, B, F, A, J, D, G]. A lower bin B precedes upper bin A. Two more reshuffling moves are required than when A precedes B. Reversely if order 2 is first picked up, bins are retrieved in a sequence of [A, J, D, G, H, C, E, B, F]. Then, lower bin D precedes upper bin C and lower bin J precedes upper bin H. Four more moves are required than when C precedes D and H precedes J. Whereas both sequences violate Principle 3.1, the former sequence less violates. From this observation, the second principle is established with the notion of violating score $v(t)$, which is the number of bin pairs violating Principle 3.1 in a bin sequence t.

Principle 3.2. *For two orders i_1 and i_2 where $v([t(i_1), t(i_2)]) < v([t(i_2), t(i_1)])$, they should be picked up in an order of (i_1, i_2).*

An optimal sequence is derived from these two principles. An optimal within-order sequence is simply a sorted list of bins by their depth. An optimal order sequence is more complicated. If each bin is demanded by only one order, violating scores are independent between order pairs. Then, an optimal sequence is obtained by sorting orders by summation of pair-wise violating scores. In practice, however, some bins (technically an SKU in it) are repeatedly demanded by many orders. Because a bin once retrieved goes top of its stack, a violating score is sequence-dependent. An optimal within-order sequence also changes accordingly. Total $q!$ sequences have to be evaluated to find the minimally violating sequence. To avoid such exponential complexity, this study suggests an efficient heuristic in the next section.

4. **Algorithm Development.**

This section develops an algorithm to derive an optimal sequence of retrieving bins following Principles 3.1 and 3.2. First, the optimal within-order sequence t_i^* of order i is trivially a sort list of its bins by their depth levels:

$$t_i^* = (j[1], j[2], \ldots, j[k_i]) \text{ where } d(j[1]) < d(j[2]) < \cdots < d(j[k_i]) \quad (1)$$

Meanwhile, an optimal between-order sequence is obtained by enumerating all $q!$ sequences and evaluating their violating scores. A greedy heuristic algorithm is developed to solve it in polynomial time. The algorithm computes violating scores of every directional pair of orders. For each order, the scores for all other orders are summed up. The summation implies a violating score induced by processing the order in the first place. Orders are sequentially selected by this score. Note that, as mentioned above, it is not exact since a bin moves to another depth after retreival. To improve precision, pair-wise violating scores are updated periodically. The detailed algorithm is as follows.

Order Sequencing Algorithm. With a set of orders $R = \{r_1, \ldots, r_q\}$, and optimal within-order sequences t_i^*'s,

1) Construct a pair-wise violating score matrix $V_{q\times q}$ of which element (i_1, i_2) is $v([t(i_1), t(i_2)])$. Its diagonal elements are 0.
2) Compute $u_i = \sum V_{i\cdot}$ for each i.
3) Choose an order set B of size b by ascending order of u_i. When two orders tie, choose smaller k_i (preferring shorter processing time).
4) From the first to the last order of B, sequentially update optimal within-order bin sequences t_i^*, $r_i \in B$, and depth level $d(j)$'s of all affected bins. Replace R with $R' = R \backslash B$ and go to step 1).
5) Repeat 1) $\sim$ 4) until no order remains.

This algorithm sequentially chooses orders from start to end. The violating score matrix V is updated for every b order during this process. The size b determines a tradeoff between precision and computation time. Larger b reduces computation time, but decreases precision, and smaller b does vice versa.

5. Numerical Study.

This section numerically evaluates performance of the developed algorithm. The warehouse specification is referred from a real-world AutoStore$^{\text{TM}}$ system: the stack grid is of size (15×48) and the stack depth is $n = 15$ levels. Assuming the grid is about a half-full, total 7,000 bins are randomly stacked in out of $15 \times 15 \times 48 =$ 10,800 slots. This example case generates $q = 500$ random orders, each of which required different numbers of bins varying from 1 to 23 bins. They are randomly assigned from an exponential distribution with mean 3. The number of bin retrievals to serve the orders is $K =$ 1,276. Total 600 unique bins are chosen out of the 7,000 bins. They are sampled 1,276 times with replacement to assign bins to orders. The most frequently retrieved bin is required by 14 different orders.

The objective function is the number of reshuffling moves. As benchmarks, randomly generated sequences and an unconstrained lower limit are compared with the algorithm solutions. The lower limit value is obtained by counting reshuffling moves independently for each bin to be retrieved. A repeatedly retrieved bin is counted only for the first retrieval assuming it stays at the top afterward. It corresponds to the summation of $d(j)$'s of unique bins at the initial state. It is practically infeasible since the roof floor buffer is highly limited and every retrieval incurs reshuffling. The computation time to get a solution is another criterion. The experiment was conducted on an Intel-i7 3.6GHz machine with 8GB memory.

Table 1 shows the results. Any feasible solution cannot achieve less than 8,150 moves by the lower limit. The best solution (8,878) comes when $b = 1$, which means a violating score matrix is updated order by order. It is only 8.9% larger than the lower limit. As b grows, a solution loses precision, but the computation time exponentially decreases.

TABLE 1. Comparison of performance

Solutions	No. of reshuffling	Computation time
Randomly generated sequences (10,000 samples)	Mean: 9,422 Std. Dev.: 43	–
Algorithm solution (all at once)	9,266	0.1 sec.
Algorithm solution $b = 50$	9,088	0.8 sec.
Algorithm solution $b = 1$	8,878	38 sec.
Unconstrained lower limit	8,150	–

The worst case is the all-at-once solution (9,266), which means that $b = q$ and an initial violating score matrix is never updated. It is 13.7% larger than the lower limit, but instantly computed. The solution quality monotonically improves for smaller b's at the cost of computation time. In practice, the system manager can choose an appropriate b according to their computational capacity. If she wants to get the best solution within a second, she can choose $b = 50$, which is 11.5% larger than the lower limit.

Meanwhile, the randomly generated sequences have a normal-like distribution with standard deviation $\sigma = 43$. The best and worst algorithm solutions (8,788 and 9,266) are 12.7σ and 3.6σ depart, respectively, from their average solution (9,422). It means that the algorithm-induced efficiency is nearly impossible to attain by an unplanned random sequence.

6. Conclusions and Future Research.

This study proposes a heuristic algorithm to determine a sequence of picking orders in a vertically stacked and top-retrieval AVS/R system. This sequence is especially useful for warehouses fulfilling retail shopping orders, each of which is composed of multiple SKU items. If an order has bins that are stored upper than all bins of another order, it should be picked up first to minimize reshuffling; it is the fundamental principle of the optimal solution. Since many orders conflict with each other in this principle, this problem needs to evaluate exponentially many sequences to find the optimal order sequence. Instead, this study presents an efficient algorithm to achieve a heuristic solution. This solution was 8.9%∼13.7% worse than the unconstrained lower limit in the numerical study.

The algorithm may be improved in both precision and computation time. Whereas the proposed one adopted a greedy approach, other heuristic approaches, like tabu search or genetic algorithm showing good performance for sequencing problems, may be promising.

Acknowledgement. This work was supported by the Ministry of Education of the Republic of Korea and the National Research Foundation of Korea (NRF-2017R1D1A1B0303 2176).

REFERENCES

[1] R. de Koster, Automated and robotic warehouses: Developments and research opportunities, *Logistics and Transport*, vol.38, pp.33-40, 2018.

[2] E. Tappia et al., Integrated storage-order picking systems: Technology, performance models, and design insights, *European Journal of Operational Research*, vol.274, no.3, pp.947-965, 2019.

[3] X. Zhao et al., Analysis of the shuttle-based storage and retrieval system, *IEEE Access*, vol.8, pp.146154-146165, 2020.

[4] G. D'Antonio and P. Chiabert, Analytical models for cycle time and throughput evaluation of multi-shuttle deep-lane AVS/RS, *The International Journal of Advanced Manufacturing Technology*, vol.104, nos.5-8, pp.1919-1936, 2019.

[5] G. D'Antonio et al., Analytical models for the evaluation of deep-lane autonomous vehicle storage and retrieval system performance, *The International Journal of Advanced Manufacturing Technology*, vol.94, nos.5-8, pp.1811-1824, 2018.

[6] A. M. M. Almassri, T. Kariya, C. Takizawa and H. Wagatsuma, A systematic evaluation method for product configurations in the dynamic warehouse focusing on the zone-specificity, *International Journal of Innovative Computing, Information and Control*, vol.16, no.4, pp.1313-1322, 2020.

[7] B. Zou, R. D. Koster and X. Xu, Operating policies in robotic compact storage and retrieval systems, *Transportation Science*, vol.52, no.4, pp.788-811, 2018.

[8] S. Tjeerdsma, *Redesign of the AutoStore Order Processing Line*, Master Thesis, Industrial Engineering and Management, University of Twente, 2019.

[9] K. H. Kim and G.-P. Hong, A heuristic rule for relocating blocks, *Computers & Operations Research*, vol.33, no.4, pp.940-954, 2006.

[10] Y. Lee and Y.-J. Lee, A heuristic for retrieving containers from a yard, *Computers & Operations Research*, vol.37, no.6, pp.1139-1147, 2010.

[11] D.-W. Jang, S. W. Kim and K. H. Kim, The optimization of mixed block stacking requiring relocations, *International Journal of Production Economics*, vol.143, no.2, pp.256-262, 2013.

[12] K. H. Kim, Y. M. Park and K.-R. Ryu, Deriving decision rules to locate export containers in container yards, *European Journal of Operational Research*, vol.124, no.1, pp.89-101, 2000.
[13] N. Boysen and S. Emde, The parallel stack loading problem to minimize blockages, *European Journal of Operational Research*, vol.249, no.2, pp.618-627, 2016.
[14] S. Boge and S. Knust, The parallel stack loading problem minimizing the number of reshuffles in the retrieval stage, *European Journal of Operational Research*, vol.280, no.3, pp.940-952, 2020.
[15] J. Lehnfeld and S. Knust, Loading, unloading and premarshalling of stacks in storage areas: Survey and classification, *European Journal of Operational Research*, vol.239, no.2, pp.297-312, 2014.