Item Assignment Problem in a Robotic Mobile Fulfillment System

Hyun-Jung Kim, Cristobal Pais, and Zuo-Jun Max Shen

***Abstract*— A robotic mobile fulfillment system (RMFS) performs the order fulfillment process by bringing inventory to workers at pick-pack-and-ship warehouses. In the RMFS, robots lift and carry shelving units, called inventory pods, from storage locations to picking stations where workers pick items off the pods and put them into shipping cartons. The robots then return the pods to the storage area and transport other pods. In this article, we consider an item assignment problem in the RMFS in order to maximize the sum of similarity values of items in each pod. We especially focus on a reoptimization heuristic to address the situation where the similarity values are altered so that a good assignment solution can be obtained quickly with the changed similarity values. A constructive heuristic algorithm for the item assignment problem is developed, and then, a reoptimization heuristic is proposed based on the constructive heuristic algorithm. Then, computational results for several instances of the problem with 10–500 items are presented. We further analyze the case for which an item type can be placed into two pods.**

***Note to Practitioners*—This article proposes an efficient heuristic algorithm for assigning items to pods in a robotic mobile fulfillment system (RMFS) so that items ordered together frequently are put into the same pod. Computational results with 10–500 items show that the gaps from upper bounds are very small on average. For cases where the similarity values between items change or their estimation is not accurate due to the fluctuations in demand, a reoptimization heuristic algorithm that alters the original assignment is developed. The experimental results show that the reoptimization algorithm is robust when perturbation levels are approximately 40%–50% of the original similarity values with much less computation times. We believe that this research work can be very helpful for operating the RMFS efficiently.**

***Index Terms*— Heuristic algorithm, item assignment problem, robotic mobile fulfillment system (RMFS), stock location problem, warehouse automation system.**

I. Introduction

IN MANY warehouses, products or items are stacked on shelves and picked and packed according to customer orders. An automated storage/retrieval system (AS/RS) is widely used when there is a high volume of loads being moved in and out of the warehouse. In some other centers, workers pick items and send them to shipping stations through a conveyor system. Traditional automation approaches for a warehouse have some drawbacks, such as the inflexibility of changing inventory locations, a high setup cost, and manual reslotting [1]. To address such issues, a robotic mobile fulfillment system (RMFS) in which robots carry shelves with items to workers was introduced [1], [2].

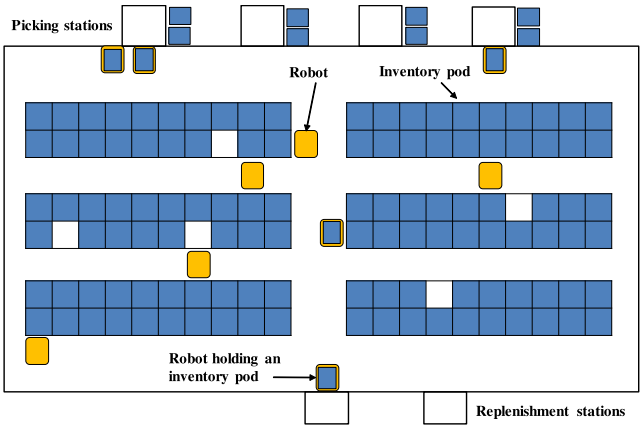


Fig. 1. System layout of the RMFS.

The RMFS targets pick-pack-and-ship warehouses. In the system, robots lift and carry three-foot-square shelving units, called inventory pods or simply pods, from storage locations to picking or replenishment stations where workers can pick items from the pods or fill the pods with items, respectively. The RMFS is typically arranged in a grid with storage zones of inventory pods, picking stations, and replenishment stations, as shown in Fig. 1. There are six inventory pod blocks, ten robots, four picking stations, and two replenishment stations in Fig. 1.

When a customer order arrives, it is assigned to one of the picking stations. Robots are then sent to specific pods that contain the items in the order and transport them to the station. Since all items are carried by robots, the operators stay in their stations and only pick items when guided by laser pointers that identify the proper items in a pod [2]. The robots wait for the workers to pick items and then return the pods to the storage location, which does not have to be the same as the previous spot. After the robots put the pods in the storage zone, they move to other pods to transport them. When a pod becomes almost empty, a robot carries it to one of the replenishment stations, and a worker in the station fills the pod

Manuscript received January 26, 2019; revised November 2, 2019; accepted January 3, 2020. Date of publication March 24, 2020; date of current version October 6, 2020. This article was recommended for publication by Associate Editor K. Harada and Editor M. P. Fanti upon evaluation of the reviewers' comments. *(Corresponding author: Hyun-Jung Kim.)*

Hyun-Jung Kim is with the Department of Industrial and Systems Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon 34141, South Korea (e-mail: hyunjungkim@kaist.ac.kr).

Cristobal Pais and Zuo-Jun Max Shen are with the Department of Industrial Engineering and Operations Research, University of California, Berkeley, CA 94720-1777 USA.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASE.2020.2979897

with items. Hence, pods are transported to either a picking or replenishment station and stored in the storage blocks between these two activities. At the picking stations, workers build shipping cartons and push them onto a take-away conveyor to transport the finished boxes to the shipping area.

One robot can carry one pod at a time, and a pod can visit one station at a time or several stations sequentially. Since pods are square or rectangular, items can be accessed from one or more of the four faces of the pods, and robots can rotate pods in order to present the correct face. Robots are bidirectional and have sensors for obstacle detection. They follow markers placed on the floor while transporting pods [2].

This RMFS has many advantages over typical warehouse automation systems, including greater accountability, adaptive slotting, and spatial flexibility [1]. Each order is completely filled by one worker, which improves accuracy and limits downstream dependence. The system easily adapts to the changes in stocking policies, and even if a robot breaks down, the system can continue to operate and it will not have a noticeable impact on productivity [1]. Moreover, the system can be expanded by simply adding more pods, robots, and stations.

When a warehouse automation system such as the RMFS is used, one of the important operation issues is to decide which items to put on which pods in order to improve the system efficiency (minimization of the robot movement distance or the number of visits of pods to picking stations). If items that many customers order at the same time are put into the same pod, it can be inferred that the number of visits of pods to picking stations can be reduced, as well as the robot movement distance. Hence, we propose a method to classify item types into groups so that each group has items that are often ordered together. To do so, some items are designated as representative ones in each group and other items, which are frequently ordered together with each representative item, are assigned to the corresponding group. The objective is to maximize the sum of the similarity values of items in each group. We especially focus on a reassignment problem of items to address the situation where the similarity values are altered so that a good assignment solution can be obtained quickly with the changed similarity values. Any item can be a representative in a group, and each item type is assigned to only one group. The items in a group are stored in one pod, and the number of item types stored in a pod is limited. In Section II, we review the related studies and then formulate the problem with a mathematical programming model in Section III. We then develop a constructive heuristic for the item assignment problem in Section IV. A reoptimization method is proposed to address the situation where the similarity values are altered after solving the original problem in Section V. Computational results for the problem with 10–500 items are presented in Section VI. We finally derive a property for items that can be put into two pods in Section VII.

II. Literature Review

The item assignment decision in the RMFS is closely related to the stock location problem that determines storage areas of items in a warehouse. Heskett [3] proposed a criterion called the cube-per-order index (COI) for the placement of stock in a distribution warehouse to minimize the labor cost associated with assembling items from stock to fill customer orders. The COI is the ratio of the item's storage space requirement (cube) to its popularity (number of storage/retrieval requests for the item). Later, Malmborg and Bhaskaran [4] showed that the COI rule, which assigns items with a large COI to the shelves closest to stations, provides an optimal solution to the storage and retrieval interleaving system.

Based on the previous studies in [3]–[5], a great deal of research has been performed on the storage assignment of correlated items in a warehouse [6]–[16]. Hwang *et al.* [6] considered a stock location problem in a low-level picker-to-part system with the objective of minimizing the workload required for order picking operations and proposed a linear programming model and compared it with the COI rule. Hwang *et al.* [13] also presented several heuristics for batching a set of orders to minimize the total distance traveled by the order picking machine. Mulvey and Crowder [14] utilized the Lagrangian relaxation to solve a clustering problem with a fixed number of clusters. Frazele and Sharp [7] formulated an item storage location problem, which is NP-hard, and solved it with a two-phase constructive heuristic. Lee [8] presented a storage assignment algorithm by considering the order structure and frequency for a man-on-board AS/RS. In that article, item similarity was computed by the items' propensity to be requested together frequently in customer orders. Xiao and Zheng [9] considered the production bill of material information for a correlated storage location assignment problem and extended the method reported in [8]. Kim [10] studied a clustering problem to assign storage locations and proposed a heuristic algorithm that minimizes the inventory-related cost and material-handling cost. Liu [11] employed a primal–dual-type algorithm to group items with a quantity-based similarity measure, and the sum of similarity in each group was maximized. Jane and Laih [15] proposed a heuristic algorithm for the item assignment in a synchronized zone order picking system, and Chiang *et al.* [12] considered a storage assignment problem for newly delivered products to reduce the travel distance. Berglund and Batta [16] proposed an analytical solution procedure for optimal placement with the probability mass function of the order pick points. Most studies have considered a typical warehouse where workers pick items and send them to packaging stations and tried to minimize the total travel time (or cost) of workers or workload of order picking operations. Hence, they mostly divide the warehouse storage area into multiple zones so that each worker takes charge of each zone. Then, the studies assign items with problem-specific or general algorithms so that each zone has items with high similarity values. They are different from the item assignment in the RMFS in which the distance between items is not important in the RMFS and the correlation between the items in a pod is much more essential. In the RMFS, robots transport items but move only one pod at a time, whereas workers pick multiple items from different shelves and send them to the packaging stations. Hence, the number of visits of pods or the robot travel distance

by considering the items in each pod should be minimized to improve the efficiency of the RMFS, and a new algorithm for such a system should be developed.

Recently, Kuo *et al.* [17] considered item assignment in the synchronized zoning system and used particle swarm optimization and a genetic algorithm to minimize the idle time of the pickers. Pan *et al.* [18] used a genetic algorithm for item assignment in pick-and-pass warehousing, and Wu and Wu [19] proposed the tabu search in synchronized zoning systems to assign items. Pang and Chan [20] presented a data mining-based algorithm for storage location problems in a randomized picker-to-parts warehouse with the objective of minimizing the total travel distances for both put-away and order picking operations. Fontana and Cavalcante [21] used the preference ranking organization method to determine the best location of items in a warehouse in order to minimize the sum of storage space cost and order picking cost. Dijkstra and Roodbergen [22] provided a dynamic programming approach for storage location assignment problems using the proposed route length formulas in a picker-to-parts warehouse. Basile *et al.* [23], [24] presented a hybrid modeling approach based on a Petri net and a simulation tool for an automated warehouse system and analyzed its performance. They showed the effectiveness of the approach with a real case study. The previous studies have mostly considered the AS/RS or synchronized zone order picking systems with multiple blocks and aisles and used various item similarity measures based on the order frequency or quantity of items. They focus on grouping items and then assign each group to a storage location with their own rules. Further details regarding the warehouse stock location problem can be found in [26] and [27], and some metaheuristic algorithms can also be used for the problem [28].

D'Andrea [25] introduced automation and robotics innovations of the RMFS and explained its challenging issues. The item assignment problem in the RMFS is closely related to the path planning of robots with collision avoidance. Many researchers have studied multirobot path planning algorithms and their complexity [29], [30]. Zou *et al.* [31] addressed an item assignment problem by considering handling speeds of stations and used a neighborhood search algorithm based on queuing networks. The results showed that a random assignment rule is outperformed by the proposed handling-speed-based assignment rule. Yuan and Gong [32] designed the RMFS with queuing network models and proposed the optimal number and velocity of robots. Lamballais *et al.* [33] also developed queuing models for the RMFS to estimate the maximum order throughput, average order cycle time, and robot utilization.

Robots in the RMFS assume the role of workers in a warehouse who pick items from the shelves and then send them to shipping stations. The workers pick multiple items from different shelves while walking along the shelves, whereas robots carry one pod at a time and hence transport only the items in the pod. Hence, the picking and travel time of the robots is only affected by the correlation between the items in each pod. The item assignment problem in the RMFS varies from those in previous studies. First, the similarity measure we consider is different. In general, the similarity value s_{ij} for items i and j is obtained by considering the frequency of the items' appearance in the same order or quantity and ranges from 0 to 1 and s_{ii} is 1. Hence, even though there is only one item in a group, its similarity value becomes 1, which may result in a large number of groups. Hence, the number of groups is limited in previous studies [11], [14]. We use the number of orders requiring a pair of items for their similarity value and set s_{ii} to 0 in this article. Putting one type of item in a pod in the RMFS does not reduce the number of visits of pods or the robot movement distance, which leads to a system efficiency drop. If two items that are often ordered together are put into a pod, workers in the picking station can pick up the two items at one visit of the pod. However, if each pod has only one type of item, two pods with the two items should be transported to the station. Even if customers order only one type of item, the system efficiency can be decreased by putting one type of item in a pod because workers in the picking stations handle multiple orders from 6 to 12 at the same time. Second, the number of groups is not restricted because there are hundreds to thousands of pods in warehouses. As more item types are put into a pod, the system efficiency can be improved in general, which reduces the number of groups of items. Hence, the number of item types that can be stored in a pod is constrained. In the RMFS, similarity values among items in different pods do not affect the performance of the system because pod storage locations change from time to time and each robot can handle one pod at a time. Hence, the similarity between the items in the same pod is an important factor that determines the throughput of the system. Therefore, the algorithm in the previous literature should be modified and extended significantly for the item assignment problem in the RMFS. In addition, there can be fluctuations in product demands depending on the season or time of day, which requires the recalculation of similarity values. Then, a new assignment solution modified from the current solution by reoptimizing the problem is required. We, hence, propose a heuristic algorithm for the item assignment problem and more focus on updating the algorithm for the cases in the presence of changes in the items' similarity values.

III. Problem Description and Model Formulation

In this article, we propose an item assignment model for the RMFS that maximizes the sum of the similarity values of items in the groups. Since the storage location of pods changes dynamically, it is not easy to estimate the total movement distance of pods. However, it is inferred that we can save the number of trips of pods to picking stations or robot movement distance by putting items that are often ordered together into the same pod. The similarity value of a pair of items in this article is the number of orders requiring both items in the pair together so that the items that are often ordered together with a representative item in each group are assigned to the group. Therefore, the objective is to maximize the number of orders requiring both a representative item and other items in each group. Pods are assumed to be empty at the beginning and filled with items instantaneously. We note that the proposed algorithm can also be used for pods that already have some items with a simple modification. The number of item types stored in a pod is restricted because, first, putting

Fig. 2. Assignment of five items.

many different types of items in each pod is not an easy job at replenishment stations and, second, we can reflect practical needs by adjusting the number of item types stored in each pod. Each item type is assigned to only one group and items in each group are stored in one pod. Suppose that we have five items and each pod can contain the maximum of two item types. When items 1 and 2, items 3 and 4, and item 5 are grouped together, the items in each group are put into the same pod as shown in Fig. 2 with pods 1–3. However, if a pod is not large enough to store the items in a group, they can be distributed in several pods. Suppose that there are 100 units of item 1 and 100 units of item 2 in a group where the sizes of the two items are similar. If a pod can contain up to 100 units, two items are put into two pods, each of which has 50 units of item 1 and 50 units of item 2 as pods 1 and 4 in Fig. 2. Due to the size of a pod, small items are considered in the RMFS in general.

The following notations are used throughout this article.

Inputs and Parameters:

1) I: Set of item types.
2) n: Number of item types.
3) μ_i: Number of orders requiring item i for each $i \in I$;
4) a_{il}: Number of orders that require both items i and l, for each $i \in I$, $l \in I$.
5) p: Maximum number of item types that can be stored in a pod.

Let A be a matrix whose components are a_{il} $\forall i, l \in I$. $a_{il} = a_{li}$ $\forall i, l \in I$, and a_{ii} $\forall i \in I$ is equal to 0. a_{il} is the similarity value between items i and l. μ_i and a_{il} are derived from demand orders for a certain period of time. We note that $\mu_i \geq \max_{l \in I} a_{il}$ but μ_i can be smaller or larger than $\sum_{l \in I} a_{il}$ because a_{il} contains not only orders that have only items i and l but also the orders that contain other items in addition to items i and l.

The decision variable is as follows: X_{ij} is 1 if item i is assigned to group j and 0 otherwise, for each $i, j \in I$. Here, group j means a group whose representative item is item j. Such an item is also called a centroid or cluster median [11]. We use representative items and centroids alternatively in this article. Items that are ordered together frequently should be put into the same pod so that workers in picking stations can pick multiple items at the same time from the pod.

Example 1: Suppose that there are four items whose μ_i values are 50, 40, 30, and 20, and A_1, the similarity matrix for this example, is as follows:

$$A_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 10 & 20 & 5 \\ 10 & 0 & 10 & 5 \\ 20 & 10 & 0 & 5 \\ 5 & 5 & 5 & 0 \end{pmatrix} \end{array}.$$

If each pod can contain two types of items, it is optimal to put item 3 with item 1 and item 4 with item 2, and the sum of similarity values in all groups is 25.

Now, we derive the formulation as follows:

$$\text{Problem P1}: \quad \max \sum_{i \in I} \sum_{j \in I} a_{ij} X_{ij} \tag{1}$$

$$\text{s.t.} \quad \sum_{j \in I} X_{ij} = 1 \quad \forall i \in I, \tag{2}$$

$$\sum_{i \in I} X_{ij} \leq p * X_{jj} \quad \forall j \in I, \tag{3}$$

$$X_{ij} \in \{0, 1\} \quad \forall i \in I \ \forall j \in I. \tag{4}$$

The objective is to maximize the sum of similarity values of items in each group. For each item i in group j, the number of orders that contain both items i and j is added. Each item is assigned to only one group according to constraint (2), and the number of item types in a group is restricted by p from constraint (3). The sum of similarity values in groups can be considered as the number of trips of pods that can be saved by putting items that are ordered together into the same pod. P1 is a clustering problem, where a set of items is divided into multiple clusters (or groups) so that the similarity values of items in each group can be maximized, with a limited number of item types in groups and no given number of groups [14], [34], [35].

The objective only considers the similarity values between a centroid and items in each group, and the similarity values between each pair of items (not a centroid) in a group are not considered. Such a problem can represent a situation well where a part of items have high demands and high similarity values with other items. Many large warehouses handle hundreds of items, but only a part of them, approximately 20%–30%, account for a large portion of sales. Such items might become a centroid of each group in our problem. Another objective of considering the sum of similarity values between each pair of items in a group can also be used. The comparison between our objective and the sum of similarity values between each pair of items can be performed as future work.

In the proposed problem, the number of representative items (or groups) should be larger than or equal to $\lceil (n/p) \rceil$ because at most p types of item can be assigned to a pod. The maximum number of groups is n. Then, the possible number of combinations for representative items is $\sum_{i=0}^{n-\lceil (n/p) \rceil} \binom{n}{n-i}$. In addition, for each group, at most $p-1$ items should be selected in order to maximize the objective function. Since orders arrive dynamically and similarity values can be changed from time to time, the assignment may often require modifications. Hence, the assignment solution should be obtained as fast as possible. We, therefore, propose a constructive heuristic and a reoptimization heuristic in Sections IV and V, respectively. We note that other general heuristic algorithms, such as a genetic algorithm, simulated annealing, or variable neighborhood search, can also be used [28], but the proposed algorithm is derived by analyzing the problem itself and its performance is analyzed in Section IV.

IV. Solution Approach

A. Heuristic Algorithm for the Item Assignment Problem

Item j with a small number of orders has small similarity values with other items in general because $a_{ij} \leq \min(\mu_i, \mu_j)$. Hence, the objective value is likely to improve if items with a large number of orders become representative items. One simple and efficient way to assign items is a greedy algorithm; assign items that have high similarity values with a representative item to the corresponding group. In example 1, we first designate item 1 as a representative item and then assign item 3 to group 1 because $a_{31} \geq a_{21}$ and $a_{31} \geq a_{41}$. Then, the two items left are grouped together, and we obtain an optimal solution. However, if $a_{32} = 25$, it is better to assign item 4 to item 1 and item 3 to item 2 because the objective is increased to 30. Hence, when assigning items to group j, we need not only to compare a_{ij} $i \in I$ but also to consider a_{il}, $l \neq j, l \in I$. By taking these observations into account, we propose a heuristic algorithm.

We first introduce some notations. Let a_i^k and $a_{(i)}^k$ indicate the kth largest value among a_{il} $\forall l \in I$ and the ith largest value among a_l^k $\forall l \in I$, respectively. The same values are ordered arbitrarily. Set V_j^k contains item i that satisfies $a_{ij} = a_i^k$. In example 1, a_1^1, a_2^1, a_3^1, and a_4^1 are 20, 10, 20, and 5, respectively. Since $a_{21} = a_2^1 (= 10)$, $a_{31} = a_3^1 (= 20)$, and $a_{41} = a_4^1 (= 5)$, V_1^1 contains items 2–4, and similarly, V_2^1 has item 4, V_3^1 has items 1, 2, and 4, and V_4^1 has no item. In example 1, $a_{(1)}^1$, $a_{(2)}^1$, $a_{(3)}^1$, and $a_{(4)}^1$, are 20, 20, 10, and 5, respectively. S_j is a set of item indices in group j, including item j. $S_j + \{i\}$ and $S_j - \{i\}$ indicate the sets of group j after adding and removing item i, respectively. Sets L and L' contain indexes of all items and representative items, respectively. In example 1, $S_1 = \{1, 3\}$, $S_2 = \{2, 4\}$, $L = \{1, 2, 3, 4\}$, and $L' = \{1, 2\}$. $C(S_j)$ indicates $\sum_{i \in S_j} a_{ij}$, and r^l is used for the representative item of a group that has the lth largest sum of similarity values, i.e., $r^1 = \arg\max_{j \in L'} \sum_{i \in S_j} a_{ij}$. In example 1, $C(S_1)$ and $C(S_2)$ are 20 and 5, respectively, and r^1 and r^2 are 1 and 2, respectively. Without loss of generality, we assume that n is a multiple of p in deriving and analyzing the algorithm by adding dummy jobs with similarity values of 0. The heuristic algorithm is as follows.

Constructive Heuristic

Step 1: Sort items in nonincreasing order of μ_i such that $\mu_1 \geq \mu_2 \geq \cdots \geq \mu_n$. $\{1, 2, \ldots, n\} \in L$ and $m \leftarrow 1$.

Step 2: Select the first item, j, from L, add it to S_j and L', and then $k \leftarrow 1$.

Step 2-1: **While** $|V_j^k| == 0$, **Do** $k \leftarrow k + 1$.

Step 2-2: **While** $|S_j| \neq p$ or $|V_j^k| \neq 0$, **Do** assign an item in L to S_j that has the largest a_{ij}, where $i \in V_j^k$. Eliminate item i from V_j^k.

Step 2-3: **If** $|S_j| \neq p$, **Then** go to Step 2-1.
Else $a_{il} \leftarrow 0$, $a_{li} \leftarrow 0$ for each $i \in S_j$ and l where $1 \leq l \leq n$, update V_b^d, where $1 \leq b, d \leq n$, and remove i's ($\in S_j$) from L.

Step 2-4: **If** $L == \emptyset$, **Then** go to Step 3.
Else go to Step 2.

Step 3: Sort groups in nonincreasing order of $C(S_i)$ where $i \in L'$. $\alpha \leftarrow 1$, and $\beta \leftarrow 2$.

Step 3-1: **For** any pair of items h and l in S_{r^α} and S_{r^β}, respectively,
If $C(S_{r^\alpha}) + C(S_{r^\beta}) \leq C(S_{r^\alpha} - \{h\} + \{l\}) + C(S_{r^\beta} - \{l\} + \{h\})$, **Then** switch them and do not consider them for further pairwise interchange.

Step 3-2: **If** $\alpha == n - 1, \beta == n$, **Then** go to Step 4.
If $\beta == n$, **Then** $\alpha \leftarrow \alpha + 1$, $\beta \leftarrow \alpha + 1$, and go to Step 3-1.
Else $\beta \leftarrow \beta + 1$ and go to Step 3-1.

Step 4: Obtain an objective value and keep the best solution so far.
If $m < n$, **Then** $m \leftarrow m + 1$, $L' \leftarrow \emptyset$, $\{m, m+1, \ldots, n, 1, \ldots, m-1\} \in L$, and go to Step 2 with the original similarity matrix A.
Else finish the algorithm.

Step 1 requires $O(n \log n)$, and step 2 can be implemented in $O(n^3 \log n)$ with $O(n^2 \log n)$ for updating V_j^k for all items, which should be repeated at most n times. Step 3 can be implemented with $O(n^2)$. Hence, the time complexity of the constructive heuristic is $O(n^4 \log n)$ because the maximum number of iterations is n.

The items are first sorted according to μ_i, and then, item 1 becomes the first representative and other items are assigned to group 1 in nonincreasing order of a_{i1}, $i \in V_1^1$ if $|V_1^1| \neq 0$. This means that the items that have the highest similarity value with item 1 are first assigned. When item i is assigned, the elements of i's row and column in A are set to 0, and V_j^k is updated for the remaining items. After assigning all of the items, a pair of items in different groups is exchanged if it improves the objective value in step 3. Step 3 ends when there are no more pairwise interchanges. Then, we repeat steps 2 and 3 with item 2, item 3, ..., item n as the first representative item at a time. We note that the original similarity matrix A in step 4 indicates the matrix that is not modified from step 2 in the constructive heuristic. As we mentioned, items in a group are assigned to one pod. The number of groups obtained from the constructive heuristic is $\lceil (n/p) \rceil$.

The two constraints we consider are that each item should be assigned once and the maximum number of items in each group is p. They are easily satisfied with the proposed algorithm. The algorithm first designates representatives with a given rule and then assigns other items to the representatives by considering p. Once an item is assigned as a representative or to a group, the related elements in a similarity matrix A become 0 and the item index is eliminated from L. Hence, it is not selected again, and a solution from the algorithm is feasible.

In the proposed algorithm, the items with a large number of orders are first designated as representative items in order. However, we repeat the procedure n times by changing the item orders in step 4 of the constructive heuristic, and hence, the items with a small number of orders can also be selected as

representatives. Since the items are sorted based on the number of orders, the proposed algorithm may behave differently for different μ_i values even if a_{ij} is the same. We later provide two more alternatives for choosing representative items with a_{ij} and compare the three rules.

Example 2: Suppose that there are four items in which μ_i values are 100, 80, 70, and 60, p is 2, and A_2, the similarity matrix for this example, is as follows:

$$A_2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 60 & 20 & 50 \\ 60 & 0 & 65 & 40 \\ 20 & 65 & 0 & 30 \\ 50 & 40 & 30 & 0 \end{pmatrix} \end{array}.$$

From the constructive heuristic, since μ_1 is the largest one, item 1 becomes the first representative, and $S_1 = \{1\}$ in step 2. V_1^1 contains only item 4 since $a_{41} = a_4^1$, and hence, item 4 is assigned to the group and $S_1 = \{1, 4\}$ in step 2-2. Then, the elements related to items 1 and 4 in A_2 become 0 and go to step 2. The next item with the large number of orders is item 2, and it is designated as the second representative item with $S_2 = \{2\}$. V_2^1 contains item 3 since $a_{32} = a_3^1$, and hence, item 3 is assigned to the representative item 2, and $S_2 = \{2, 3\}$ in step 2-2. The objective is 115. No pairwise interchange is required. The first solution is kept, and then, item 2 becomes the first representative. $a_{12} = a_1^1$ and $a_{32} = a_3^1$, but $a_{32} > a_{12}$. Hence, item 3 is assigned to item 2. Item 4 is then chosen as the next representative item, and item 1 is assigned again. The procedure is repeated until item 4 becomes the first representative.

The worst case bound of the constructive heuristic is analyzed in the Appendix.

B. Heuristic Algorithm Initialization Extensions

In the original heuristic approach, representatives are chosen as the items that have the highest demand among available ones. In order to explore different initialization rules, we propose two methods for selecting representative items based on the similarity matrix A.

1) Sort by Average a_{ij} Values: The first extension is to sort the items i by their average values of a_{ij}. The idea behind this is that items with higher $\hat{A}_i = \sum_{j \in I}(a_{ij}/(n-1))$ might tend to be better representatives since their potential similarity values with respect to other items can be larger than that of items with smaller average values.

Step 1: Sort items in nonincreasing order of $\hat{A}_i = \sum_{j \in I}(a_{ij}/(n-1))$ such that $\hat{A}_1 \geq \hat{A}_2 \geq \cdots \geq \hat{A}_n$.

New representatives would be obtained in comparison to the ones generated using the original approach. Note that in this case, the relation between the value of $\hat{A}_i$ and the objective value is closer than the one obtained based on demand levels μ_i.

2) Sort by $\max a_i^k$ *Values:* In the second extension, items are sorted by their maximum values $a_i^{k=1}$, $\forall i \in I$, and designated as representative items in order. This method generates groups with centroids that tend to have the maximum similarity values in the A matrix so that the chances of obtaining better initial groups are increased.

Step 1: Sort items in nonincreasing order of $a_i^{k=1} = \max_{j \in I} a_{ij}$ such that $a_1^{k=1} \geq a_2^{k=1} \geq \cdots \geq a_n^{k=1}$.

V. Reoptimization Heuristic Algorithm: Similarity Matrix Change

Even if a solution (called an initial or original solution) is obtained from the A matrix, it might be necessary to modify the solution, especially when similarity values in the A matrix are recalculated for the next period or when the values in A are not as accurate as the decision-maker wants and are recalculated after obtaining the initial solution. This leads us to rerunning the constructive heuristic or taking advantage of the current solution by reoptimizing the problem from this point. Since products would have some fluctuations in their demands depending on the season or time of day, it would be interesting to check how the solution changes depending on the level of modifications in the A matrix.

In this section, we provide an extension to the constructive heuristic that reoptimizes the initial solution by considering the new similarity values. We let A and A' be the original similarity matrix and newly obtained matrix modified from A, respectively. We would be able to compare the performance of the reoptimization approach with the heuristic for different levels of n and p.

A. Reoptimization Heuristic Algorithm

In the reoptimization algorithm, each pair of items is classified into five cases depending on the type of items involved: 1) a centroid and an item in the same group; 2) a centroid and an item in a different group; 3) a centroid and another centroid; 4) two items in the same group; and 5) two items in different groups. For each case, a pairwise interchange procedure that exchanges a pair of items in a different group is applied to the original solution without performing as many steps as in the constructive heuristic so that the better performance in terms of the computation time is obtained. This method is suitable, especially when dealing with a large number of items. We note that the performance of the reoptimization algorithm is significantly affected by the level of change of A' in comparison to the original matrix A since larger discrepancies between the two matrices would lead to very different solutions.

The main objective of the reoptimization heuristic is to obtain a good-and-fast solution of the altered A' matrix based on the initial solution from the original matrix A without generating a new solution directly from the altered A' matrix.

B. Heuristic Implementation

The main strategy of the algorithm is to classify the differences between both matrices A and A' and performing a series of pairwise interchange procedures in order to explore new solutions that would improve the objective function. We have an initial improvement step (case 0) and have five cases (cases 1–5) depending on the item types as follows.

1) Case 0—Initial Improvement: The initial step of the reoptimization heuristic is to switch centroids of groups in the original solution to improve the objective value. The following rule is executed at the beginning of the algorithm and after each case below is applied.

1) Take item l^*, where $l^* = \arg\max_{l \in S_i} \sum_{j \in S_i} (a_{jl}/(|S_i| - 1))$ for each representative group with centroid i. Replace centroid i to centroid l^*.

2) Case 1—Similarity Changed Between Centroid c and Item i in the Same Group: In this case, there are two main possibilities.

1) *Similarity Increase:* Since centroid c and item i are in the same group, the current solution is improved due to the increase in the similarity value a_{ic}, and thus, it is still the best possible solution. The objective value is recalculated.
2) *Similarity Decrease:* Since the similarity value between the centroid c and item i in its group decreases, it may be better to exchange items i and j in a different group or, if there are groups having items less than p, move item i to one of these groups.
   a) Item i is exchanged by item j from a different group (not a centroid), where item j, in the group with centroid l, maximizes the value of $a_{il} - a_{jl}$.
   b) Item i is exchanged by item j from a different group (not a centroid), where item j, in the group with centroid l, maximizes the value of $a_{il} + a_{jc}$.
   c) Item i is exchanged by item j from a different group (not a centroid), where item j maximizes a_{jc}.
   d) If there are sparse groups that have items less than p, item i is moved to one of these groups. No item is added to the group containing centroid c. The sparse group, which maximizes a_{il} where item l is the centroid of the group, is selected.

3) Case 2—Similarity Changed Between Centroid c and Item i in Different Groups:

1) *Similarity Increase:* It can be better to move item i to the group with centroid c or exchange centroid c with the centroid, item l, of the group containing item i.
   a) Item i is exchanged by item j (not a centroid), where item j, in the group with centroid c, minimizes a_{jc}.
   b) Item i is exchanged by item j (not a centroid), where item j, in the group with centroid c, maximizes a_{jl}.
   c) Centroid c is exchanged by centroid l from the group containing item i.
   d) When the group containing item i is a sparse group, centroid c is moved to the group and replaces its centroid l.
2) *Similarity Decrease:* Since the centroid and item are in different groups, the objective value is not affected, and thus, we keep the current solution and no exchange is applied.

4) Case 3—Similarity Changed Between Centroids c and l in Different Groups:

1) *Similarity Increase:* There can be a situation where changing centroid c (or l) with an item in the group containing centroid l (or c) could lead to an improvement in the objective value.
   a) Item i, which has the lowest a_{il} in the group with centroid l, is exchanged with centroid c.
   b) Item j^*, where $j^* = \arg\max_{j \in S_c} ((\sum_{i \in S_l} a_{ij})/|S_l|)$, is exchanged with centroid l.
   c) Item i, in the group with centroid l, is exchanged with centroid c. Item i then becomes the new centroid of the group. Repeat this with k items, the number of exchanges allowed, i.e., 2–5 in this case, from the group with centroid l and keep the best solution obtained.
   d) If the group containing l as the centroid is sparse, centroid c is moved to this group (or vice versa if S_c is sparse).
2) *Similarity Decrease:* As in the previous case, no exchange is performed since the similarity value between centroids does not affect the objective value, and no improvement will be obtained.

5) Case 4—Similarity Changed Between Items i and j in Different Groups: In this case, it would be possible to obtain an improvement by changing the compositions of both groups, designating one of the items as the centroid of the other group.

1) *Similarity Increase:* Two main exchange rules can be applied in order to make an improvement without exploring the entire solution space.
   a) Item i is exchanged by centroid c in the group containing item j. Item i is the new centroid of the group containing item j.
   b) Item j is exchanged by centroid l in the group containing item i. Item j is the new centroid of the group containing item i.
2) *Similarity Decrease:* Since both items are in different groups, we keep the current solution as the best one without performing any exchange.

6) Case 5—Similarity Changed Between Items i and j in the Same Group:

1) *Similarity Increase:* In this case, we have two main possibilities of using item i or item j as a centroid of their current group such that an improvement is achieved.
   a) Item i is selected as the centroid of its current group.
   b) Item j is selected as the centroid of its current group.
2) *Similarity Decrease:* This is an irrelevant case since the objective value is not going to be improved if the similarity of two items inside the same group is decreased. No exchange rules are applied.

Based on the previous description, the algorithm is applied as follows.

Reoptimization Heuristic

Step 1: Given a solution, OrigSol, from the original similarity matrix A, and the altered A' matrix, perform the procedure of Case 0 and obtain a new solution, NewSol.

Step 2: Perform the procedure of Case 1 to NewSol for all pairs of items corresponding to Case 1, and then apply the procedure of Case 0. Let NewSol1 be a solution obtained after this step.

Step 3: Perform the procedure of Case 2 to NewSol1 for all pairs of items corresponding to Case 2, and then apply the procedure of Case 0. Let NewSol2 be a solution obtained after this step.

Step 4: Perform the procedure of Case 3 to NewSol2 for all pairs of items corresponding to Case 3, and then apply the procedure of Case 0. Let NewSol3 be a solution obtained after this step.

Step 5: Perform the procedure of Case 4 to NewSol3 for all pairs of items corresponding to Case 4, and then apply the procedure of Case 0. Let NewSol4 be a solution obtained after this step.

Step 6: Perform the procedure of Case 5 to NewSol4 for all pairs of items corresponding to Case 5, and then apply the procedure of Case 0. Let NewSol5 be a solution obtained after this step.

NewSol5 is the solution obtained with the reoptimization heuristic.

VI. Computation Results

A. Heuristic Algorithm Versus Optimal Solutions

We compare the sum of similarity values obtained from the constructive heuristic to the upper bound or an optimal solution to verify its effectiveness. P1 is solved with ILOG CPLEX v12.7.1 for 10 min. The solution from the constructive heuristic is compared to an optimal value if it is obtained within 10 min, and otherwise, the best upper bound found for 10 min is used for the comparison. The number of orders, μ_i, is generated randomly between (100, 300). $a_{il}\ \forall i, l \in I$ is generated as follows. Items are sorted based on the demand. Then, for item 1, the similarity values with other items are generated randomly while satisfying $0 \le a_{1l} \le \min(\mu_1, \mu_l)$, where $2 \le l \le n$. For item 2, a_{2l}, where $3 \le l \le n$, is obtained randomly between $\max\{a_{12} + a_{1l} - \mu_1, 0\}$ and $\min(\mu_2, \mu_l)$. For item 3, a_{3l}, where $4 \le l \le n$, is obtained between $\max\{a_{13} + a_{1l} - \mu_1, a_{23} + a_{2l} - \mu_2, 0\}$ and $\min(\mu_3, \mu_l)$. In a similar way, a_{ij}, where $i + 1 \le j \le n$, is randomly generated between $\max\{a_{1i} + a_{1j} - \mu_1, a_{2i} + a_{2j} - \mu_2, \ldots, a_{i-1,i} + a_{i-1,j} - \mu_{i-1}, 0\}$ and $\min(\mu_i, \mu_j)$. Suppose that a_{12} and a_{13} are 50 and 60, respectively, with μ_1 of 100. This implies that at least ten orders require both items 2 and 3. Hence, a_{23} should be generated between 10 and $\min(\mu_2, \mu_3)$.

The proposed algorithm is implemented in the 64-bit version of Python 3.6 on a personal computer with a 2.4-GHz four-core I7-4700 CPU and 12.00-GB RAM using Windows 10 as the main OS. The experimental setting of the algorithm is as follows.

1) A series of instances is generated with μ_i for all $i \in I$ where $100 \le \mu_i \le 300$. The relevant parameters are: 1) $n \in \{10, 20, 30, 40, 50, 100, 200, 300, 400, 500\}$ and 2) $p \in \{2, 3, 4, 5, 10, 15, 20, 50, 100, 250\}$ where $p \le (n/2)$.
2) For each possible combination of (n, p), 100 instances are generated to capture the real performance of the algorithms. The average values are presented in the tables and graphs.

TABLE I

Computation Results for the Item Assignment Problem With μ_i From (100, 300): Small Instances

n	p	Gap from Upper Bounds [%]	Running Time [s]
10	2	0.67	0.002
	3	4.54	0.002
	4	3.51	0.003
	5	3.89	0.003
20	2	1.54	0.015
	3	4.78	0.018
	4	4.43	0.016
	5	4.33	0.018
	10	5.01	0.020
30	2	2.21	0.043
	3	4.95	0.056
	4	5.01	0.062
	5	5.08	0.064
	10	4.67	0.066
	15	6.22	0.064
40	2	2.53	0.114
	3	4.25	0.138
	4	4.56	0.152
	5	5.00	0.161
	10	4.31	0.163
	15	4.45	0.159
	20	5.22	0.165
50	2	2.53	0.248
	3	4.33	0.367
	4	4.54	0.583
	5	5.03	0.521
	10	4.46	0.430
	15	3.90	0.472
	20	4.87	0.465
	25	5.49	0.463
Overall Result		4.21	0.17

Tables I–III show the experimental results with 10–500 items for μ_i from (100, 300), and a different number of item types stored in a pod, p. Table IV summarizes the results. The "gap from upper bounds" is obtained by (solution from CPLEX for 10 min − objective value from the constructive heuristic) × 100/solution from CPLEX for 10 min. The average running time of the constructive heuristic can be found in the fourth column.

The gap tends to decrease as n becomes large. The maximum gap among all instances was obtained from an instance with 200 items and p of 100 in Table II, which is 11.24%. The instances with 100 items are solved within 1 s, whereas it takes approximately 15 and 80 s for instances with 300 and 500 items, respectively. For instances with 1000 items, it takes 15 min to obtain a solution from the constructive heuristic. All the instances with more than 400 items require approximately 1 h to obtain an optimal solution with CPLEX. We note that as p increases, the objective value also increases because more item types can be stored and the number of groups decreases. However, the difference between objective values with p of 5 and 10 is much larger than that with p of 15 and 20. This means that the objective values top out at a certain point

TABLE II

COMPUTATION RESULTS FOR THE ITEM ASSIGNMENT PROBLEM WITH μ_i FROM (100, 300): MEDIUM INSTANCES

n	p	Gap from Upper Bounds [%]	Running Time [s]
100	2	0.82	0.75
	3	1.33	0.88
	4	1.01	0.74
	5	1.54	0.79
	10	1.88	0.75
	15	1.91	0.99
	20	1.94	0.94
	25	2.45	0.84
	50	2.84	0.88
200	2	2.23	4.89
	3	2.55	5.33
	4	2.22	5.48
	5	3.44	4.97
	10	3.16	4.56
	15	1.89	5.99
	20	1.54	6.45
	25	2.68	6.23
	50	8.45	6.48
	100	11.24	6.23
300	2	1.44	15.44
	3	2.39	15.32
	4	2.53	14.94
	5	2.47	14.84
	10	1.39	14.76
	15	1.07	15.04
	20	0.60	15.06
	25	0.55	14.74
	50	2.16	14.89
	100	2.22	14.77
	150	8.34	14.11
Overall Result		2.68	7.60

TABLE III

COMPUTATION RESULTS FOR THE ITEM ASSIGNMENT PROBLEM WITH μ_i FROM (100, 300): LARGE INSTANCES

n	p	Gap from Upper Bounds [%]	Running Time [s]
400	2	2.39	50.45
	3	2.03	48.56
	4	2.74	49.66
	5	2.41	51.20
	10	1.45	44.15
	15	1.46	44.98
	20	1.69	42.52
	25	1.84	55.12
	50	1.99	53.26
	100	2.04	49.74
	150	2.44	47.70
	200	4.23	46.50
500	2	2.20	94.86
	3	1.06	93.12
	4	3.49	84.23
	5	2.50	72.46
	10	3.06	70.91
	15	2.81	70.45
	20	3.25	69.45
	25	2.80	72.42
	50	1.34	73.96
	100	1.17	68.65
	150	2.92	70.44
	200	1.54	81.26
	250	1.84	84.24
Overall Result		2.27	63.61

because items that are rarely ordered with a representative one are added with large p.

We conducted experiments for instances with 10–50 items because the gap was large when n and p are small, as shown in Table I. We can observe that the average gap is increased but is still less than 4.5% on average. It is notable that it takes a long time to obtain an optimal solution with CPLEX when n is not a multiple of p because some groups have items less than p (sparse groups).

TABLE IV

SUMMARY ON AVERAGE GAP AND RUNNING TIME WITH n

n	Gap from Upper Bounds [%]	Running Time [s]
10	3.15	0.00
20	4.02	0.02
30	4.69	0.06
40	4.33	0.15
50	4.39	0.44
100	1.75	0.84
200	3.94	5.66
300	2.29	14.91
400	2.23	48.65
500	2.31	77.42
Overall Result	3.31	14.82

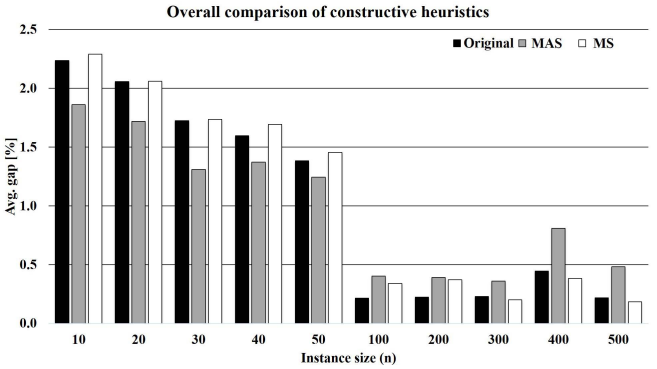


Fig. 3. Overall results for the three constructive heuristic methods.

The proposed algorithm is applied when there are empty pods and items that should be stored. It can also be used for the pods with some space for more items even if they already have other types of items. In this case, the number of items that can be stored in pods is different, and hence, p is changed to p_j for group j. The constructive heuristic can still be applied by replacing p with p_j in steps 2-2 and 2-3.

B. Heuristic Algorithm Extensions

The two newly proposed approaches in Section IV-B, maximum average similarity (MAS) and maximum similarity (MS) value for selecting the representative items of each group, are tested and compared with the performance of the original rule. All three algorithms are used to construct an initial solution to the problem. Average gaps with respect to the best objective value reached per iteration are reported across all the replications of each instance. The best objective value per iteration indicates the maximum sum of similarity values obtained by any of the three approaches. The number of orders, μ_i, is generated randomly between 100 and 300.

We separate the analysis into two main approaches: 1) an overall summary comparing the average gaps obtained for each instance size n and 2) a specific analysis for comparing the performance of the three implemented heuristic rules depending on the maximum number of different items, p, that can be selected for each inventory pod.

1) Overall Results: In Fig. 3, we compare the average gap values obtained for each rule, original, MAS, and MS rules. Based on the result, we can easily see the following patterns.

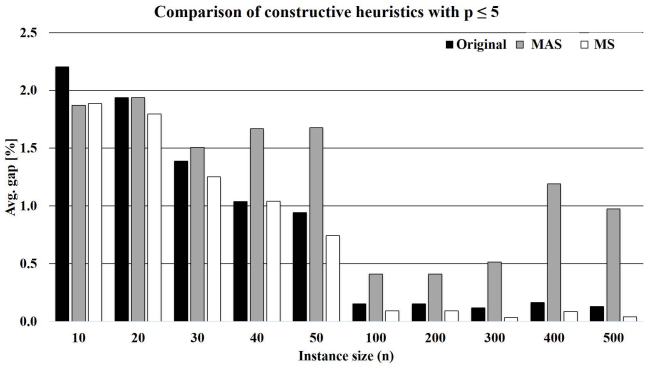


Fig. 4. Computational results for the three constructive heuristic methods with $p \leq 5$.

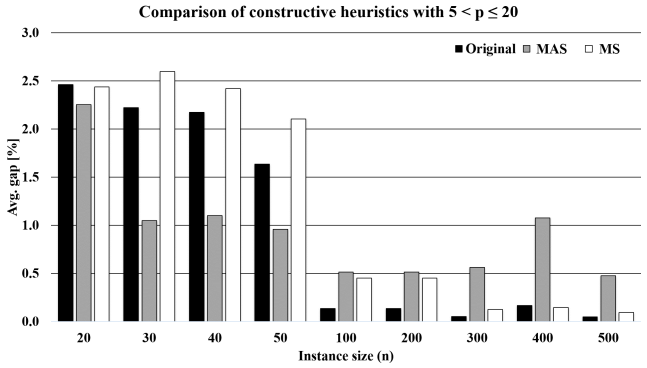


Fig. 5. Computational results for the three constructive heuristic methods with $5 < p \leq 20$.

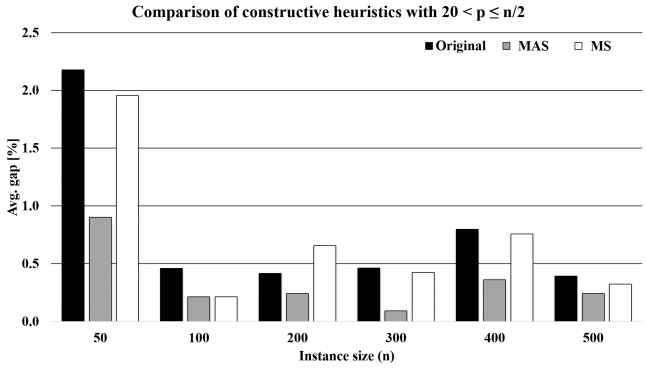


Fig. 6. Computational results for the three constructive heuristic methods with $20 < p \leq (n/2)$.

1) The maximum average rule obtains the best performance among all the options in the instances, including up to 50 items.
2) After this threshold, the original algorithm, as well as the MS, obtains similar results, while the maximum average rule tends to be outperformed (i.e., worst average gap).

Small instances (up to 50 items) are likely to have different similarity values, and thus, taking the items with the MAS values has significant effects on the objective value. As there are more items, the similarity value difference among items tends to be small and, thus, the maximum average value rule loses its impact on the objective value and obtains worse results than other rules.

2) Specific Results by Different p Ranges: As mentioned earlier, we want to understand and discover the most suitable heuristic rule depending on the characteristics of an instance. Thus, a specific analysis based on the value of p is performed for all instances.

1) $p \leq 5$: In Fig. 4, we can notice how the performance of the MAS rule is outperformed by the other approaches. Since the number of items in each pod is very limited, the maximum values are more important than the average values in terms of the impact on the objective. Differences up to 1.5% are encountered between the worst rule and the others. We can see a clear tendency where the best selection rule for these instances is the MS one. The solutions are approximately 0.25% better than the original approach and not significant for the n tested.
2) $5 < p \leq 20$: In Fig. 5, the pattern is exactly the opposite to the previous range of p values for instances with at most 50 items. The best performance is obtained by the maximum average rule that reaches the differences up to 1.5% in comparison to the other rules. After this n threshold, the performance of the original algorithm is the best one among the rules. However, differences are not significant with the MS rule.
3) $20 < p \leq (n/2)$: In Fig. 6, the best performance is reached by the MAS rule, which confirms our preliminary analysis. The average tends to increase its impact as more items are added to the pods. Average differences of approximately 0.5% for instances with more than 100 items are obtained.

In order to test the rules for more general cases, as well as to check the scalability of the implemented rules, a series of massive instances with $n \in \{1000,\ 2000, \ldots, 10\,000\}$ was tested. A similar pattern was obtained, but differences tend to be more significant (approximately 3%) in terms of the average gaps reported.

C. Reoptimization Heuristic

We compare the performance of the original constructive heuristic using A' as the similarity matrix and the proposed reoptimization algorithm starting from the initial solution obtained from the A matrix. Both the running times and final gaps with respect to the optimal (or best bound) solution are compared for the generated instances. In order to obtain relevant insights from the results, we separate the analysis into two main sections: 1) a summary comparing the average gaps and running times obtained for each instance size n and 2) a specific analysis for comparing the performance depending on p for each inventory pod. The similarity value, a_{il}, is changed to $a_{il} + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$ and σ is randomly selected between 40% and 50% of the average similarity value. We note that larger changes in matrix A lead to poor reoptimization results.

1) Overall Results: In Table V, we can see summary results by comparing the average gap values and running

TABLE V
AVERAGE COMPUTATION RESULTS FOR ORIGINAL AND REOPTIMIZATION HEURISTICS

Instance	Gap from Upper Bounds [%]		Running Time [s]		
n	Original	Re-Opt	Original	Re-Opt	Orginal/Re-Opt
10	3.15	**2.68**	0.01	**0.01**	1.0
20	4.02	**3.74**	0.02	**0.01**	2.6
30	4.69	**4.38**	0.06	**0.02**	2.7
40	4.33	**3.73**	0.15	**0.06**	2.4
50	4.39	**4.12**	0.44	**0.16**	2.7
100	2.01	**1.90**	0.84	**0.12**	6.7
200	3.94	**3.64**	5.66	**0.51**	11.1
300	2.30	**1.90**	14.91	**1.01**	14.8
400	2.23	**2.20**	48.65	**2.97**	16.4
500	2.31	**2.22**	77.42	**4.24**	18.3
Overall Results	3.34	**3.05**	14.82	**0.91**	7.9

times obtained by the original constructive and reoptimization algorithms. Based on the results, we can see the following patterns.

1) Small instances (up to 50 items) tend to reach a worse average gap than larger instances. The average gap values for these instances vary from 1.90% ($n = 100$ with the reoptimization technique) up to 4.69% in the worst case ($n = 30$ with the constructive heuristic). In contrast, larger instances tend to converge to the average gap obtained, approximately 2.37% ($= (1.90 + 3.64 + 1.90 + 2.20 + 2.22)/5$), with a clear outlier case for $n = 200$.
2) The reoptimization algorithm tends to outperform the original constructive heuristic among all instances with differences up to 0.5% in the overall average gap values.

Small instances (up to 50 items) are very easily solved by direct optimization algorithms (e.g., CPLEX). Hence, the optimal solutions are easily obtained by solving the formulation, and thus, we compare the heuristic solutions with the global optimal objective ones. However, for larger instances where CPLEX is not able to find optimal solutions within the imposed time limit, a comparison with respect to the best upper bound is performed. For large instances with 500 items, the upper bound gap obtained with CPLEX in 10 min is about 5%.

To study the potential impact on the computation time, we calculate the ratio of running times of the original and reoptimization methods in Table V. From the results, the ratio goes up to 18 with an average of 7.9 across all instance sizes. The ratio in terms of saved time follows an increasing pattern with n. Hence, the reoptimization approach does not have a significant impact when dealing with small instances (up to 50 items), while it becomes the most suitable approach for larger instances, justifying its implementation for the tested altered level.

2) Results by Assortment Level p: For completeness, an analysis by different values of p is performed among all instance sizes n. As can be seen in Fig. 7, the average gap of both heuristics tends to be very similar for all assortment p levels. Figs. 8–10 show the average gaps for each p level where $p \leq 5$, $5 < p \leq 20$, and $20 < p \leq (n/2)$, respectively. As p becomes large, the average gap of the reoptimization heuristic tends to be smaller than that of the constructive one. In Fig. 8, the gaps with $n \geq 100$ have smaller values than

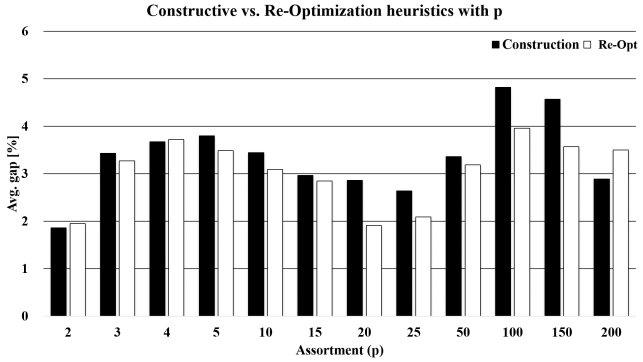


Fig. 7. Computational results for the average gap with different values of p.

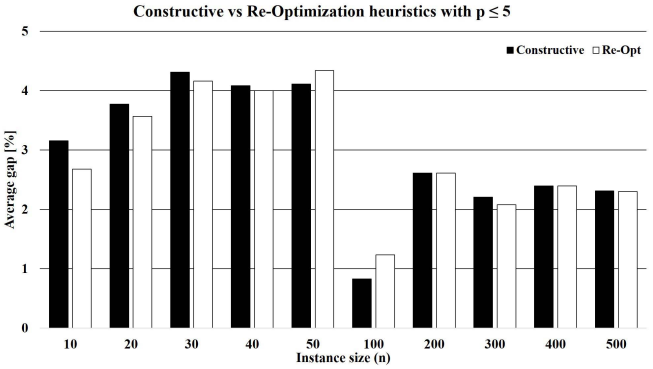


Fig. 8. Computational results for the average gap with $p \leq 5$.

those with $n < 100$, and the total average gaps of the constructive and reoptimization techniques are 2.98% and 2.94%, respectively. In Fig. 9 with $5 < p \leq 20$, we can also see the trend of small gaps with large n, and the difference between average gaps of constructive and reoptimization heuristics (3.25%, 2.85%) becomes large compared to those in Fig. 8. When n is small, the reoptimization heuristic performs better than the constructive one in general. The difference between the two heuristics is further increased as p becomes large, as shown in Fig. 10. The total average gaps of constructive and reoptimization heuristics are 4.61% and 3.89%, respectively, and hence, the difference is 0.72. The reoptimization heuristic performs better than the constructive one regardless of n in Fig. 10. Overall, the difference between the two algorithms is not large as can be seen in Fig. 7, showing us the stability of the proposed reoptimization algorithm. This fact is very

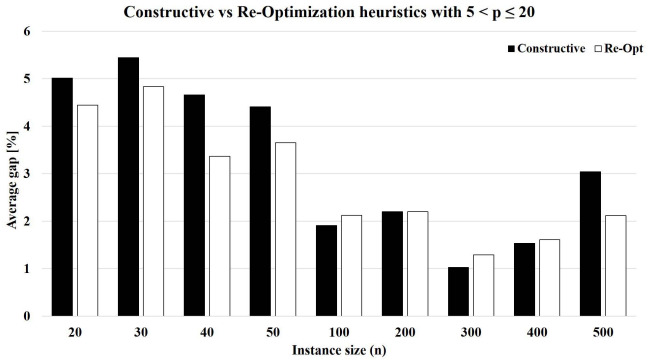


Fig. 9. Computational results for the average gap with $5 < p \leq 20$.

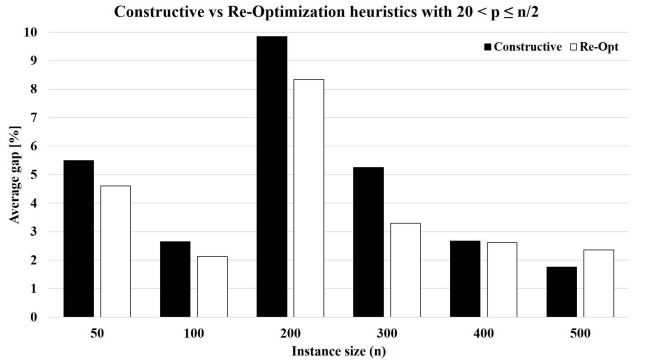


Fig. 10. Computational results for the average gap with $20 < p \leq (n/2)$.

important because it indicates that the reoptimization heuristic is robust for different sizes and p levels when change levels of similarity values are approximately 50% of the original ones, and thus, a new solution is not needed for finding relatively high-quality solutions.

However, a formal study regarding the level of change in A should be performed for determining a cutoff value that will help the decision-maker decide which approach to use: 1) create a new solution and apply an improvement heuristic for optimizing it or 2) use the current solution available as a starting point and reoptimize it.

Based on the results, it can be clearly concluded that the reoptimization approach is successful for the current perturbation level in terms of running times and average gap values reached.

VII. Discussion: An Item Type in Two Pods

In Sections III–VI, it is assumed that one item type is put into only one pod. A typical warehouse has three days' worth of inventory, and hence, one-third of the inventory should be refilled each day [2]. If we consider the replenishment activity, some items with a high number of orders should be assigned to more than two pods so that while one pod with the item is replenished, the other one can serve customer orders. Hence, we assume that some item types can be put into two pods when the interarrival time of their orders is far less than the replenishment time, and such items ($\in I'$) are given.

We need to determine the portion of customer orders of item i, μ_{i_1} and μ_{i_2} ($\mu_{i_1} + \mu_{i_2} = \mu_i$) that pods containing item i handle. Item i can also be considered as two different items i_1 and i_2 with μ_{i_1} and μ_{i_2} orders, respectively. One pod may serve almost all of the orders, and the other one may be used only while refilling items in the first pod. Therefore, there is a limit L_i ($\mu_{i_1}, \mu_{i_2} \geq L_i$) for each item i, and L_i refers to the order rate during the replenishment activity. Then, μ_{i_1} and μ_{i_2} become decision variables to be determined. For the items that can be put into two pods, we derive the following property.

Proposition 1: When item i is stored in two pods, the number of orders that the pods handle, μ_{i_1} and μ_{i_2}, should be $\mu_i - L_i$ and L_i, respectively, to maximize the sum of similarity values.

Proof: Assume that item i is divided into i'_1 with μ'_{i1} and i'_2 with μ'_{i2}, where $\mu'_{i_1} \leq \mu_{i_1}$ $(= \mu_i - L_i)$, $\mu'_{i_2} \geq \mu_{i_2}$ $(= L_i)$, and $\mu'_{i_1} \geq \mu'_{i_2}$. We show that it is better to assign customer orders of item i to two pods with as many as $\mu_i - L_i$ and L_i than μ'_{i_1} and μ'_{i_2}, respectively. In the case of one item in each pod, there is no difference because $\mu_{i_1} + \mu_{i_2} = \mu'_{i_1} + \mu'_{i_2} = \mu_i$. Suppose that item i is put with item q in one pod and with item r in another pod, and $a_{iq} \geq a_{ir}$ without loss of generality. To improve the objective value, item q is put into the pod with item i_1, and item r is put into another pod with item i_2. Hence, $a_{i_1q} - a_{i'_1q} = a_{iq}((\mu_{i1} - \mu'_{i1})/\mu_i)$ and $a_{i'_2r} - a_{i_2r} = a_{ir}((\mu'_{i2} - \mu_{i2})/\mu_i)$. Then, $a_{i_1q} - a'_{i_1q} \geq a'_{i_2r} - a_{i_2r}$ because $a_{iq} \geq a_{ir}$ and $\mu_{i_1} + \mu_{i_2} = \mu'_{i_1} + \mu'_{i_2} = \mu_i$. Therefore, $a_{i_1q} + a_{i_2r} \geq a'_{i_1q} + a'_{i_2r}$, and the sum of similarity values taking care of μ_{i_1} and μ_{i_2} orders is larger than that with μ'_{i_1} and μ'_{i_2}. ■

We can use the proposed algorithms by simply considering item i as two different items, i_1 and i_2, with $\mu_i - L_i$ and L_i orders, respectively.

VIII. Conclusion and Future Work

In this article, the item assignment problem of the RMFS has been examined. The proposed model divides the items into several groups by considering how often an item is ordered together with the representative item. We have developed the constructive heuristic algorithm and analyzed its worst case performance bound. We then proposed a reoptimization method for the case in which the similarity values are changed. The computation results of the algorithm have shown that the gap from the upper bound is small. We have further analyzed the case for which some item types can be put into two pods.

The general case in which an item type can be assigned to more than two pods should be further analyzed. In this case, constraint (2) should be modified with $\geq$ instead of $=$. Then, the items with high similarity are assigned to multiple groups, whereas others may be assigned to only one group. Hence, we need to restrict the (minimum or maximum) number of pods that contains a certain item type, which should be determined in advance. In addition, the objective function should be modified to consider items that have different representatives. In our problem, each item type has only one representative item, whereas each can have multiple different representatives in the generalized problem. Hence, the objective function, which indicates the sum of orders requiring both an item and the representative item in a group, should be modified by considering the ratio of items' orders split into multiple groups. These issues should also be handled in the proposed

algorithm. A tight worst case bound, which does not depend on the input data, should also be analyzed. We leave this to be explored in future work.

The algorithm we provided can be applied to a wide range of other stock location problems by considering a pod as a large shelf that contains multiple types of items. The formulation can also be extended with a data-driven approach for estimating a_{il}. Another objective of adding all of the similarity values between each pair of items in a group can be considered as well. Other operational issues in the RMFS, such as robot path planning, the pod storage location problem, and the order batching problem, should be addressed as well.

APPENDIX

The effectiveness of the constructive heuristic is analyzed by providing the largest gap between the lower and upper bounds on the sum of similarity values. We let N and N^* denote the objective value from the constructive heuristic and the optimal value, respectively.

Lemma 1: An upper bound on the sum of similarity values from the constructive heuristic is obtained by $\sum_{i=1}^{n-\lceil (n/p)\rceil} a_{(i)}^1$.

Proof: $\lceil (n/p)\rceil$ items are chosen as a representative with a given p value from the constructive heuristic, and $n-\lceil (n/p)\rceil$ items are assigned to groups and contribute to maximizing the objective function. Hence, we take the largest value, $a_{(i)}^1$, for each i, $1 \le i \le n - \lceil (n/p)\rceil$. ∎

Since it is hard to numerically know how effective steps 3 and 4 of the constructive heuristic are, an initial solution is considered to compute the lower bound.

Lemma 2: A lower bound on the sum of similarity values from the constructive heuristic is obtained by $\sum_{i=1}^{\lceil (n/p)\rceil}\sum_{j=1}^{p-1} a_{(j)}^{n-i}$.

Proof: From the algorithm, $n-\lceil (n/p)\rceil$ items are assigned to $\lceil (n/p)\rceil$ groups, and items with large similarity values are first assigned to items with high demand. The worst case happens when item q is designated as a representative, but other items have the smallest similarity values with item q, i.e., $a_{iq} = a_i^{n-1}$, $i \in I$. The proposed algorithm still tries to take large values among them and assign $p-1$ items with $a_{(1)}^{n-1}, a_{(2)}^{n-1}, \ldots, a_{(p-1)}^{n-1}$ to the group. In a similar way, items with $a_{(1)}^{n-2}, a_{(2)}^{n-2}, \ldots, a_{(p-1)}^{n-2}$ may be assigned to the next group. Hence, the lower bound becomes $\sum_{i=1}^{\lceil (n/p)\rceil}\sum_{j=1}^{p-1} a_{(j)}^{n-i}$. ∎

Lemma 3: The sum of similarity values from the constructive heuristic is larger than or equal to $N^* - \sum_{i=1}^{n-\lceil (n/p)\rceil} a_{(i)}^1 + \sum_{i=1}^{\lceil (n/p)\rceil}\sum_{j=1}^{p-1} a_{(j)}^{n-i}$.

Proof: From Lemmas 1 and 2

$$\sum_{i=1}^{\lceil \frac{n}{p}\rceil}\sum_{j=1}^{p-1} a_{(j)}^{n-i} \le N \le N^* \le \sum_{i=1}^{n-\lceil \frac{n}{p}\rceil} a_{(i)}^1.$$

Hence

$$N \ge N^* - \sum_{i=1}^{n-\lceil \frac{n}{p}\rceil} a_{(i)}^1 + \sum_{i=1}^{\lceil \frac{n}{p}\rceil}\sum_{j=1}^{p-1} a_{(j)}^{n-i}.$$

∎

Theorem 1: $N \ge k_1 N^*$, in which k_1 is the largest real number that satisfies $k_1 \le (\sum_{i=1}^{\lceil (n/p)\rceil}\sum_{j=1}^{p-1} a_{(j)}^{n-i})/(\sum_{i=1}^{n-\lceil (n/p)\rceil} a_{(i)}^1)$.

Proof: From Lemma 3, if

$$-\sum_{i=1}^{n-\lceil \frac{n}{p}\rceil} a_{(i)}^1 + \sum_{i=1}^{\lceil \frac{n}{p}\rceil}\sum_{j=1}^{p-1} a_{(j)}^{n-i} \ge (k_1-1)\sum_{i=1}^{n-\lceil \frac{n}{p}\rceil} a_{(i)}^1$$

$$N \ge N^* - \sum_{i=1}^{n-\lceil \frac{n}{p}\rceil} a_{(i)}^1 + \sum_{i=1}^{\lceil \frac{n}{p}\rceil}\sum_{j=1}^{p-1} a_{(j)}^{n-i}$$

$$\ge N^* + (k_1-1)\sum_{i=1}^{n-\lceil \frac{n}{p}\rceil} a_{(i)}^1 \ge k_1 N^*$$

which leads to $N \ge k_1 N^*$

$$-\sum_{i=1}^{n-\lceil \frac{n}{p}\rceil} a_{(i)}^1 + \sum_{i=1}^{\lceil \frac{n}{p}\rceil}\sum_{j=1}^{p-1} a_{(j)}^{n-i} \ge (k_1-1)\sum_{i=1}^{n-\lceil \frac{n}{p}\rceil} a_{(i)}^1$$

is rearranged to

$$k_1 \le \frac{\sum_{i=1}^{\lceil \frac{n}{p}\rceil}\sum_{j=1}^{p-1} a_{(j)}^{n-i}}{\sum_{i=1}^{n-\lceil \frac{n}{p}\rceil} a_{(i)}^1}.$$

∎

Since the pairwise interchange effect is not considered in computing the lower bound, the largest gap is smaller than the one provided in Theorem 1. k_1 when p is 2 depends on the difference between a_i^1 and $a_{(1)}^{n-i}$, where $1 \le i \le (n/2)$. If n is an even number with p of 2, since $a_{(1)}^1 \ge a_{(2)}^1 \ge \cdots \ge a_{((n/2))}^1$ and $a_{(1)}^{n-1} \le a_{(1)}^{n-2} \le \cdots \le a_{(1)}^{(n/2)}$, k_1 can be estimated with the difference between $a_{(1)}^1$ and $a_{(1)}^{n-1}$. If

$$\frac{a_{(1)}^{n-1}}{a_{(1)}^1} = \frac{1}{k_2}, \quad k_1 = \frac{a_{(1)}^{n-1} + a_{(1)}^{n-2} + \cdots + a_{(1)}^{\frac{n}{2}}}{a_{(1)}^1 + a_{(2)}^1 + \cdots + a_{(\frac{n}{2})}^1}$$

$$\ge \frac{a_{(1)}^{n-1} + a_{(1)}^{n-2} + \cdots + a_{(1)}^{\frac{n}{2}}}{k_2 a_{(1)}^{n-1} + k_2 a_{(1)}^{n-1} + \cdots + k_2 a_{(1)}^{n-1}}$$

$$\ge \frac{a_{(1)}^{n-1} + a_{(1)}^{n-1} + \cdots + a_{(1)}^{n-1}}{k_2 a_{(1)}^{n-1} + k_2 a_{(1)}^{n-1} + \cdots + k_2 a_{(1)}^{n-1}} = \frac{1}{k_2}.$$

Hence, $k_1 \ge (1/k_2)$. If the ratio of $a_{(1)}^{n-1}$ to $a_{(1)}^1$ is $(1/3)$, then k_1 is as large as $(1/3)$. k_1 can be larger when the algorithm is applied to items with high demand and similarity values.

Lemma 1 is for an upper bound on the sum of similarity values, and Lemma 2 is for the lower bound obtained from the proposed algorithm. Then, in Lemma 3, the largest gap between the lower and upper bounds on the sum of similarity values of items in groups is obtained, and finally, in Theorem 1, the worst case performance ratio k_1 is specified. The algorithm is usually applied first to a portion of items with high similarity values and then another portion of items in practice [11].

REFERENCES

[1] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI Mag.*, vol. 29, no. 1, pp. 9–20, 2008.

[2] J. J. Enright and P. R. Wurman, "Optimization and coordinated autonomy in mobile fulfillment systems," in *Proc. 25th AAAI Conf. Artif. Intell. (AAAI)* 2011, pp. 33–38.

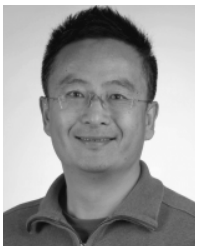
[3] J. L. Heskett, "Cube-per-order index-a key to warehouse stock location," *Transp. Distrib. Manage.*, vol. 3, no. 1, pp. 27–31, 1963.
[4] C. J. Malmborg and K. Bhaskaran, "A revised proof of optimality for the cube-per-order index rule for stored item location," *Appl. Math. Model.*, vol. 14, no. 2, pp. 87–95, Feb. 1990.
[5] H. G. Wilson, "Order quantity, product popularity, and the location of stock in a warehouse," *AIIE Trans.*, vol. 9, no. 3, pp. 230–237, 1977.
[6] H. Hwang, Y. Hui Oh, and C. Nam Cha, "A stock location rule for a low level Picker-To-Part system," *Eng. Optim.*, vol. 35, no. 3, pp. 285–295, Jun. 2003.
[7] E. A. Frazele and G. P. Sharp, "Correlated assignment strategy can improve any order-picking operation," *Ind. Eng.*, vol. 21, no. 4, pp. 33–37, 1989.
[8] M.-K. Lee, "A storage assignment policy in a man-on-board automated storage/retrieval system," *Int. J. Prod. Res.*, vol. 30, no. 10, pp. 2281–2292, Oct. 1992.
[9] J. Xiao and L. Zheng, "A correlated storage location assignment problem in a single-block-multi-aisles warehouse considering BOM information," *Int. J. Prod. Res.*, vol. 48, no. 5, pp. 1321–1338, Mar. 2010.
[10] K. H. Kim, "A joint determination of storage locations and space requirements for correlated items in a miniload automated storage-retrieval system," *Int. J. Prod. Res.*, vol. 31, no. 11, pp. 2649–2659, Nov. 1993.
[11] C.-M. Liu, "Clustering techniques for stock location and order-picking in a distribution center," *Comput. Oper. Res.*, vol. 26, nos. 10–11, pp. 989–1002, Sep. 1999.
[12] D. M.-H. Chiang, C.-P. Lin, and M.-C. Chen, "The adaptive approach for storage assignment by mining data of warehouse management system for distribution centres," *Enterprise Inf. Syst.*, vol. 5, no. 2, pp. 219–234, May 2011.
[13] H. Hwang, W. J. Baek, and M.-K. Lee, "Clustering algorithms for order picking in an automated storage and retrieval system," *Int. J. Prod. Res.*, vol. 26, no. 2, pp. 189–201, Feb. 1988.
[14] J. M. Mulvey and H. P. Crowder, "Cluster analysis: An application of Lagrangian relaxation," *Manage. Sci.*, vol. 25, no. 4, pp. 329–340, 1979.
[15] C.-C. Jane and Y.-W. Laih, "A clustering algorithm for item assignment in a synchronized zone order picking system," *Eur. J. Oper. Res.*, vol. 166, no. 2, pp. 489–496, Oct. 2005.
[16] P. Berglund and R. Batta, "Optimal placement of warehouse cross-aisles in a picker-to-part warehouse with class-based storage," *IIE Trans.*, vol. 44, no. 2, pp. 107–120, Feb. 2012.
[17] R. J. Kuo, P. H. Kuo, Y. R. Chen, and F. E. Zulvia, "Application of metaheuristics-based clustering algorithm to item assignment in a synchronized zone order picking system," *Appl. Soft Comput.*, vol. 46, pp. 143–150, Sep. 2016.
[18] J. C.-H. Pan, P.-H. Shih, M.-H. Wu, and J.-H. Lin, "A storage assignment heuristic method based on genetic algorithm for a pick-and-pass warehousing system," *Comput. Ind. Eng.*, vol. 81, pp. 1–13, Mar. 2015.
[19] Y. Wu and Y. Wu, "Taboo search algorithm for item assignment in synchronized zone automated order picking system," *Chin. J. Mech. Eng.*, vol. 27, no. 4, pp. 860–866, Jul. 2014.
[20] K.-W. Pang and H.-L. Chan, "Data mining-based algorithm for storage location assignment in a randomised warehouse," *Int. J. Prod. Res.*, vol. 55, no. 14, pp. 4035–4052, Jul. 2017.
[21] M. E. Fontana and C. A. V. Cavalcante, "Use of promethee method to determine the best alternative for warehouse storage location assignment," *Int. J. Adv. Manuf. Technol.*, vol. 70, nos. 9–12, pp. 1615–1624, Feb. 2014.
[22] A. S. Dijkstra and K. J. Roodbergen, "Exact route-length formulas and a storage location assignment heuristic for picker-to-parts warehouses," *Transp. Res. E, Logistics Transp. Rev.*, vol. 102, pp. 38–59, Jun. 2017.
[23] F. Basile, P. Chiacchio, and J. Coppola, "A hybrid model of complex automated warehouse systems—Part I: Modeling and simulation," *IEEE Trans. Automat. Sci. Eng.*, vol. 9, no. 4, pp. 640–653, Oct. 2012.
[24] F. Basile, P. Chiacchio, and J. Coppola, "A hybrid model of complex automated warehouse systems—Part II: Analysis and experimental results," *IEEE Trans. Automat. Sci. Eng.*, vol. 9, no. 4, pp. 654–668, Oct. 2012.
[25] R. D'Andrea, "Guest editorial: A revolution in the warehouse: A retrospective on kiva systems and the grand challenges ahead," *IEEE Trans. Autom. Sci. Eng.*, vol. 9, no. 4, pp. 638–639, Oct. 2012.
[26] G. Cormier and E. A. Gunn, "A review of warehouse models," *Eur. J. Oper. Res.*, vol. 58, no. 1, pp. 3–13, Apr. 1992.
[27] R. de Koster, T. Le-Duc, and K. J. Roodbergen, "Design and control of warehouse order picking: A literature review," *Eur. J. Oper. Res.*, vol. 182, no. 2, pp. 481–501, Oct. 2007.
[28] K. Gao, Z. Cao, L. Zhang, Z. Chen, Y. Han, and Q. Pan, "A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems," *IEEE/CAA J. Automat. Sinica*, vol. 6, no. 4, pp. 904–916, Jul. 2019.
[29] J. Yu and S. M. LaValle, "Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics," *IEEE Trans. Robot.*, vol. 32, no. 5, pp. 1163–1177, Oct. 2016.
[30] J. Banfi, N. Basilico, and F. Amigoni, "Intractability of time-optimal multirobot path planning on 2D grid graphs with holes," *IEEE Robot. Automat. Lett.*, vol. 2, no. 4, pp. 1941–1947, Oct. 2017.
[31] B. Zou, Y. Gong, X. Xu, and Z. Yuan, "Assignment rules in robotic mobile fulfilment systems for online retailers," *Int. J. Prod. Res.*, vol. 55, no. 20, pp. 6175–6192, 2017.
[32] Z. Yuan and Y. Y. Gong, "Bot-In-Time delivery for robotic mobile fulfillment systems," *IEEE Trans. Eng. Manag.*, vol. 64, no. 1, pp. 83–93, Feb. 2017.
[33] T. Lamballais, D. Roy, and M. B. M. De Koster, "Estimating performance in a robotic mobile fulfillment system," *Eur. J. Oper. Res.*, vol. 256, no. 3, pp. 976–990, Feb. 2017.
[34] J. M. Mulvey and M. P. Beck, "Solving capacitated clustering problems," *Eur. J. Oper. Res.*, vol. 18, no. 3, pp. 339–348, Dec. 1984.
[35] G. Cornuejols, M. L. Fisher, and G. L. Nemhauser, "Location of Bank accounts to optimize float: An analytic study of exact and approximate algorithms," *Manage. Sci.*, vol. 23, no. 8, pp. 789–810, 1977.

Hyun-Jung Kim received the Ph.D. degree in industrial and systems engineering from the Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2013.

She is currently an Assistant Professor with the Department of Industrial and Systems Engineering, Korea Advanced Institute of Science and Technology. Her research interests include modeling, scheduling, and control of production systems.

Cristobal Pais is currently pursuing the Ph.D. degree with the Department of Industrial Engineering and Operations Research, University of California at Berkeley, Berkeley, CA, USA.

His research interests include stochastic programming and optimization techniques for large size problems.



Zuo-Jun Max Shen received the Ph.D. degree from Northwestern University, Evanston, IL, USA, in 2000.

He is the Chancellors Professor with the Department of Industrial Engineering and Operations Research and also with the Department of Civil and Environmental Engineering, University of California at Berkeley, Berkeley, CA, USA. He has been active in integrated supply chain design and management, applied optimization, and decision making with limited information.