Joint Order Batching and Picker Manhattan Routing Problem

Chun-Cheng Lin[a,*], Jia-Rong Kang[a], Chung-Chih Hou[a], Chen-Yang Cheng[b]

[a] *Department of Industrial Engineering and Management, National Chiao Tung University, Hsinchu 300, Taiwan*
[b] *Department of Industrial Engineering and Management, National Taipei University of Technology, Taipei 106, Taiwan*

Abstract

In picking product items in a warehouse to fulfill customer orders, a practical way is to classify similar orders as the same batch and then to plan the optimal picker routing when picking each batch of items. Different from the previous problems, this work investigates the joint order batching and picker Manhattan routing problem, which simultaneously determines the optimal order batching allocation and the shortest picker Manhattan routing that cannot pass through storage shelves in the warehouse, under some practical constraints. This work further addresses this problem by particle swarm optimization with bad experience to avoid bad solutions, in which a novel solution representation is designed for simultaneously handling both order batching and picker routing. The idea of the design is to transform the warehouse floorplan into a grid, in which virtual *order center* and *batch center* are defined to represent symbolic positions of orders and batches of the solution, respectively. By calculating the distance between the two centers, similar orders are categorized as the same batch. Additionally, theoretical analysis of convergence and stability of the proposed approach is also derived. Performance of this approach is evaluated via comprehensive experimental analysis and a case study.

Keywords: Order picking, order batching, particle swarm optimization, logistics, warehouse management

*Corresponding author. Phone: +886-3-5731758. Fax: +886-3-5729101.
Email address: cclin321@nctu.edu.tw (Chun-Cheng Lin)

1. Introduction

Order picking is a core operation of warehouse management, based on needs of customer orders to quickly and accurately pick out the ordered product items from product shelves or other locations and move them with a specific procedure. The order picking process consists of information collection of order picking, traveling or moving and picking, as well as sorting and accumulation. A good planning for order picking can help substantially improve efficiency of warehouse management, and hence, it is of interest and importance to appropriately plan the picker routing distance as short as possible to save picker routing time and further to reduce time cost. In practice, customer orders in some industries (e.g., retailing and customized assembly) are often of great diversity and of small quantity of items. To reduce unnecessary repetitive picker routings, the orders for items with similar picking routing are generally categorized as the same batch to be picked before planning the order picking routes. A suitable *order batching* helps shorten the picker routing distance and reduce the time cost.

In practice, order-picking vehicles are used for picking and moving product items inside a warehouse for higher efficiency, and they are electric in modern warehouses. Design of the routing for order-picking vehicles in warehouses (a.k.a., *order picking routing* or *picker routing*) determines the total picker routing distance, which could be viewed as a vehicle routing problem (VRP). Lots of works on designing metaheuristic algorithms for various VRPs existed, e.g., [19, 21, 32]. For designing particle swarm optimization (PSO) approaches for VRPs, the works in [1, 2, 15] proposed the PSO with novel solution representations to solve various VRPs. The work in [8] proposed a hybrid discrete PSO for the VRP with simultaneous pickup and delivery. The work in [23] proposed a hybrid PSO for VRP. The work in [22] showed that PSO is the best approach to the order picking routing problem as compared with genetic algorithm (GA) and ant colony optimization (ACO). Therefore, PSO is suitable for the order picking routing problem for order-picking vehicles.

Recent works have considered the *joint order batching and picker routing problem* [31], which classifies customer orders into batches and then bases the batching allocation to determine the order picking routing. The work in [18] proposed a two-stage algorithm for the joint problem, which uses a cluster algorithm to allocate order batching according to similarity of orders, and then uses tabu search to plan the shortest picker routing. The work in

[29] proposed a two-stage GA to solve the joint problem; and the previous work in [4] proposed a two-stage method based on PSO and ACO to solve the joint problem. However, most of the previous methods tended to provide a solution for either the order batching [12] or the order picking routing [22], or used a two-stage approach [4, 18, 29] to first determine the order batching allocation and then decide the order picking routing. In calculating the picker routing distance, they adopted the Euclidean distance to measure the shortest distance between two locations in a warehouse, and did not consider that some locations in the warehouse cannot be passed through in practice, so that the picker routing distance under this constraint may be different from the actual warehouse environment. To meet practical situations, this work investigates the *joint order batching and picker Manhattan routing problem*, and the major difference from previous problems is to adopt the Manhattan distance (i.e., strictly horizontal or vertical path) that cannot pass through any product storage shelves in the warehouse to measure the shortest picker routing distance between two locations in the warehouse. The objective of this problem is to find the optimal batching of orders with diversified items and the shortest picker Manhattan routing in the warehouse under some practical constraints. Inheriting from the original problem in [18], this problem is generally NP-hard, i.e., it cannot be solved deterministically in polynomial time.

This work solves the concerned problem by an improved PSO (ImPSO for short) based on [14], which additionally introduces each particle's previous bad experience to accelerate the convergence process of the PSO algorithm. One of the key designs in this work is to propose a novel solution representation for ImPSO, which can simultaneously determine both order batching and picker routing. The idea behind this solution representation is to consider the warehouse floorplan as a grid and to define virtual *order center* and *batch center* as the symbolic positions of orders and batches in the warehouse, respectively. By calculating the distance between the two centers, the orders with a high degree of similarity (in terms of the distance between the two centers) are categorized as the same batch.

The main contributions of this work are given as follows:

- The problem of concern in this work is more general than the previous ones. In calculating the picker routing distance, this work considers the fact that some locations in the warehouse cannot be passed through. Additionally, as compared with the previous work in [18] restricted

to a fixed warehouse floorplan, this work presents a generalized distance calculation algorithm for different warehouse floorplans, so that the proposed approach can be applied to different warehouse configurations. Another main difference from the previous works is that the picker routing distance in this work is measured precisely by the Manhattan distance that cannot pass through any storage shelves in the warehouse.

- A novel solution representation is designed for determining both the order batching allocation and the picker Manhattan routing simultaneously.

- The proposed approach is applicable to customer orders with diversified items and small quantities, which conform to practical situations in some industries (e.g., retailing and customized assembly).

- This work theoretically analyzes stability and convergence of the proposed ImPSO algorithm. After many experiments under different parameters are conducted, it is discovered that bad experiences of particles in the ImPSO algorithm serve as an indispensable factor in the optimization process in handling the concerned problem.

The remainder of this work is organized as follows. Section 2 introduces the order picking system in warehouses, and gives a detail literature review. Section 3 describes the problem concerned in this work, proposes the algorithm with a novel solution representation for the problem, and derives a theoretical analysis for convergence and stability of the proposed algorithm. Section 4 gives a comprehensive experimental analysis of the proposed algorithm, and then compares the performance of a case study using the proposed approach and the current practice. Section 5 compares the performance of a case study using the proposed approach and the current practice. Lastly, Section 6 gives a conclusion with future work.

2. Preliminaries

This section first introduces the order picking system in warehouses, and then describes the concerned problem.

2.1. The Order Picking System in Warehouses

The purpose of the order picking system is to accurately and efficiently collect the product items ordered by customers in a warehouse [13]. Some literature surveys can be found in [6, 10, 30]. The system process of the whole order picking system can be divided into three stages: *information collection of order picking*, *traveling/moving and picking*, as well as *sorting and accumulation*. First, *information collection of order picking* refers to collection of the information on the customer orders to be handled for each batch, and on what items to be picked and moved for each batch. Second, *traveling/moving and picking* refer to all the activities of how the picker moves inside the warehouse according to the items to be picked to fill orders, confirms types and quantity of the product items to be picked when standing in front of the storage shelf, picks up the items, and moves them to the next storage shelf location or the product-collecting location. The time spent at this stage is the major time consumed for completing the order picking process. Third, *sorting and accumulation* refer to the process of collecting, sorting, and packing items to fill orders after the picker has taken all items of the orders. That is, the time required for the whole order picking system includes: the time for collecting the information for picking, the time for the overall picker routing movement, the time consumed for locating the correct storage shelf after moving close to the storage shelf, the time for picking up and confirming the right type and the right quantity of the item, and the time consumed for sorting and accumulation [29]. If the total picker routing distance is minimized, the picker routing time is minimized, and in turn, efficiency of the order picking operation could be increased.

Applying different order picking strategies and methods leads to a different total picker routing distance. Depending on whether each customer order could be split, there are two order picking strategies: *order splitting* and *order batching* [10]. The *order splitting* strategy splits an order into multiple subsidiary orders that are handed over to different pickers to pick items. After pickers complete picking all items of their own subsidiary orders, the items are sorted and accumulated at the product-collecting location. The *order batching* strategy does not split orders but classifies all customer orders into multiple batches. This strategy is further divided into four categories: the *total batching* strategy allocates order batching according to the accumulated quality of each item in different orders; *batching with a time window* allocates order batching according to the accumulated number of orders within a fixed time period [18]; *fixed-number batching* allocates order batching every time

when the orders to be picked are accumulated to a specific quantity [11]; *intelligent batching* allocates the orders with similar picking routes to the same batch [29].

After determining order batching allocation, four order picking methods are usually used for picking items in batches: *single-order-picking*, *batch picking*, *order collecting and organizing*, and *composite picking*. In the *single-order-picking* method [18], the picker completes picking all items of a single order in a batch, and then picks the other orders in this batch. This method has the advantage of simplicity and short lead time for order processing, and hence, it is easy to be applied, more flexible for rush orders, and suitable for picking the orders with large quantity and small diversity. The disadvantage of this method is that the picking route may become longer for an order with more diversified items. Additionally, the picking routes for the orders with less quantity and more diversity may be repeated many times so that the order picking is more time-consuming. The *batch picking* method first calculates the total quantity of each item in each batch, and then picks all the quantity of each item in this batch during the picker routing [7]. This method is suitable for picking a huge number of orders with less quantity. However, the orders handled by this method cannot reflect the change of orders in a timely manner, and the whole picking process tends to be complicated. The *order collecting and organizing* method integrates the orders from a specific region into a super order that is picked at one time. This method is suitable for the situation where each order in a day has only one item. The *composite picking* method is composition of single-order-picking and batch picking.

Based on the above classification, the order picking strategy adopted in this work is the *intelligent batching* strategy, in which the order batching allocation is determined according to similarity of orders and proximity of picking routes; the order picking method adopted in this work is the *batch picking* method, which combines multiple orders into the same batch and proceeds the picking operation after the total quantity of the same item of multiple orders in each batch is calculated.

2.2. Problem Description

The concerned problem is to classify customer orders into multiple batches and then to determine the picker Manhattan routing when moving product items in each batch in the warehouse, so that the total picker routing distance is as minimal as possible(which helps reduce picker routing time and increase order picking efficiency).

Consider a warehouse floorplan represented as a grid (consisting of squares) with an arbitrary boundary contour, e.g., see the warehouse grid with a rectangular contour in Figure 1, in which the black square is a *product-collecting location*; each white square is a *route location*; each gray square is a *storage location*. Note that the warehouse floorplan is not necessarily the same as Figure 1; this work can solve the warehouse grid with any boundary contour and any number of storage locations.

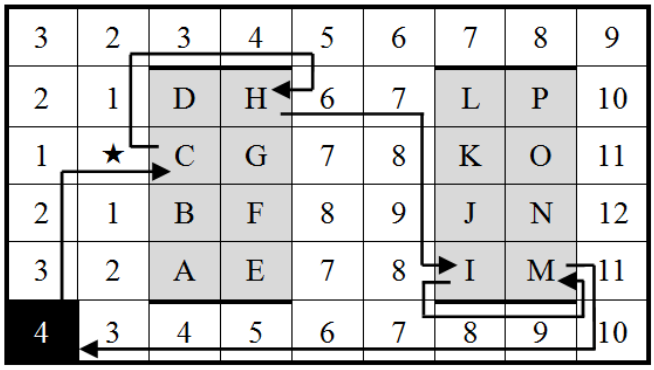

Figure 1: Illustration of a rectangular warehouse floorplan.

The *product-collecting location* is the starting point for the picker to pick items for each batch and the destination after finishing picking all items of each batch. During the picker routing, the picker can only pass through *route locations*, but cannot pass through any *storage location*. Each *storage location* has only a storage shelf, and the letter attached to each storage location in Figure 1 denotes the ID of the stored item. Each stored item is associated with a *pick-up location*, which is a white square in the grid where the picker stands in front of the storage location of the item to be picked. A bold line on the margin of a gray square in Figure 1 indicates the side from which the item cannot be picked up. For example, the square labeled by a star in Figure 1 is the pick-up location of item C, and there is a bold line on the top of squares D and H, which refers to that items D and H cannot be picked up from the top side.

This work considers the following assumptions:

- Storage locations are fixed in the warehouse. Hence, the distance between every two locations in the warehouse is measured by its Manhattan distance that cannot pass through any storage locations, and is

fixed. For example, in Figure 1, if intending to travel from the pick-up location of item A to that of item E, the picker has to take the route on the lower edge of storage locations A and E, so the shortest distance that is required actually is 5 units. If a specific customer order with items C, H, I and M is considered, the shortest picking route is the routing path shown in Figure 1.

- Each storage location accommodates only one item type; and items of the same type are not stored in different storage locations.
- The items contained in each order may be of different type, and the quantity of each item type is not limited.
- Orders are classified into multiple batches to be picked. Picking all items in each batch must be finished in a single picking route that starts and ends at the product-collecting location. Additionally, items of the same order cannot be split into different batches, i.e., picking all the items in each order must be finished in its allocated batch.
- Consider only a picker with only an order-picking vehicle. The loading capacity for the order-picking vehicle is limited, and hence, the picker cannot finish picking all orders at one time, i.e., picking must be accomplished in multiple batches. For simplicity, this work assumes that number of orders in each batch cannot exceed the maximum loading capacity of the order-picking vehicle.

With the above description for the order picking operation in a warehouse, this work attempts to propose an intelligent algorithm to determine order batching and picker routing so that the total picker Manhattan routing distance is minimized. By analog from [18], we create the mathematical model for the concerned order batching problem (in which the picker Manhattan routing is not expressed mathematically). The notation used in the problem model is as follows:

- B is set of batches;
- K is set of orders;
- L is set of storage locations;
- C is capacity of the order-picking vehicle;

- d_{ij} is the picker Manhattan routing distance between storage locations i and j under constraint of storage locations;
- S is a subset of storage locations;
- $s_{ik} = \begin{cases} 1, & \text{if an item in order } k \text{ is picked from storage location } i; \\ 0, & \text{otherwise.} \end{cases}$

Decision variables are given as follows:

- $X_k^b = \begin{cases} 1, & \text{if order } k \text{ is assigned to batch } b; \\ 0, & \text{otherwise.} \end{cases}$
- $Y_{ij}^b = \begin{cases} 1, & \text{if storage location } i \text{ is visited directly after storage} \\ & \text{location } j \text{ is visited in batch } b; \\ 0, & \text{otherwise.} \end{cases}$
- $Z_i^b = \begin{cases} 1, & \text{if storage location } i \text{ is visited in batch } b; \\ 0, & \text{otherwise.} \end{cases}$

The mathematical model is detailed as follows.

$$\text{Minimize} \quad \sum_{b\in B}\sum_{i\in L}\sum_{j\in L, j\neq i} d_{ij}Y_{ij}^b \tag{1}$$

s.t.

$$\sum_{j\in L, j\neq i} Y_{ij}^b = Z_i^b, \qquad \forall b \in B, i \in L \tag{2}$$

$$\sum_{i\in L, i\neq j} Y_{ij}^b = Z_j^b, \qquad \forall b \in B, j \in L \tag{3}$$

$$\sum_{i\in S, j\in L\setminus S} Y_{ij}^b \geq Z_i^b, \qquad \forall b \in B, S \subset L \tag{4}$$

$$Z_i^b \geq s_{ik} \cdot X_k^b, \qquad \forall b \in B, i \in L, k \in K \tag{5}$$

$$\sum_{b\in B} X_k^b = 1, \qquad \forall k \in K \tag{6}$$

$$\sum_{k\in K} X_k^b \leq C, \qquad \forall b \in B \tag{7}$$

$$X_k^b, Y_{ij}^b, Z_i^b \in \{0,1\}, \qquad \forall i, j \in L, b \in B, k \in K \tag{8}$$

where Objective (1) is to minimize the total picker Manhattan routing distance; Constraint (2) and (3) enforce that each storage location can only be the starting point or the destination once in each batch in the sense that each storage location can be passed by only once for each batch; Constraint (4) aims to prevent any routing with subsidiary loops; Constraint (5) enforces that each order k in batch b can pass by a storage location i once; Constraint (6) enforces that each order must be assigned to a particular batch; Constraint (7) enforces that the number of orders to be picked for each batch cannot exceed the maximum loading capacity C of the order-picking vehicle; Constraint (8) enforces each decision variable to be binary.

The differences between this order batching problem and the original problem in [18] are listed as follows. First, the picker routing distance d_{ij} between two storage locations is measured by the Manhattan distance that cannot pass through any storage locations, which could earnestly present the actual picker routing distance. That is, the proposed approach with such a distance calculation scheme can be applied to arbitrary warehouse configurations (any-size grid with any boundary contour); while the previous work just computed the Euclidean distance between two points in the warehouse. Second, number of orders for each batch cannot exceed the maximum loading capacity C of the vehicle. If applying the loading capacity setting of the previous model, which counts all items in each order, then the order splitting is required in the optimization process, so that the picker would face much difficulty in handling orders.

3. The Proposed Approach

This section first explains main components of the proposed approach, and then derives its algorithm analysis.

3.1. Solution Encoding

This subsection proposes a novel solution representation in the proposed PSO algorithm for the joint order picking and picker Manhattan routing problem. Consider a warehouse floorplan represented as a grid (e.g., Figure 1). Given a number of orders each of which contains multiple items, the concerned problem is to classify these orders into multiple batches, and then to find the shortest picker Manhattan routing for picking all items in each batch. Note that each batch corresponds to a picker routing that begins and ends at the product-collecting location. Our proposed approach to this

problem contains two aspects: first, each given order (with multiple items) determines a symbolic location called *order center* on the warehouse floorplan based on the storage locations of the items contained in the order in the warehouse; second, the proposed approach determines a symbolic location of each batch on the warehouse floorplan, which is called *batch center*. The distance between the two centers is used to determine similarity of orders, and further used to allocate order batching. The details are stated in the following.

Since each order contains multiple items that are spread in different storage locations of the warehouse, this work uses a single symbolic location called *order center* (a square in the warehouse floorplan grid) to represent the location of an order with multiple items so as to easily allocate order batching. The *order center* of an order is defined as a white route square with the shortest distance to the storage location of each item of this order in the warehouse. In Figure 1, for example, consider an order with items F, H, J, and L. The order center of this order is one of the two white squares between squares G and K. If multiple candidate squares for an order center exist, the square close to the product-collecting location is chosen as the center in this work. Hence, the white square close to square G is chosen as the order center in this example. As for implementation, given a problem instance, the problem setting includes information of all items of each order and the storage locations of all items in the warehouse. After calculating all the distances between storage locations and route locations, the order center of each order can be calculated.

The concerned order batching problem is to classify a number of customer orders into multiple batches, each of which corresponds to a picker Manhattan routing that begins and ends at the product-collecting location. Since each customer order is associated with an order center on the warehouse floorplan, each batch is also associated with a *batch center* which represents a symbolic location for the picker Manhattan routing formed by the batch on the warehouse floorplan. Supposing that the batch center of each batch has been known, the Euclidean distance between the order center of each customer order and the batch center of each batch is calculated, and then each customer order is allocated to the batch with the shortest distance between their centers. By doing so, the orders in the same batch would have a higher degree of similarity in terms of the distance between the order center and the batch center, so that repetitive routes for different batches can be avoided. After the order batching is allocated, the total picker Manhattan

routing distance can further be calculated according to the order batching.

In summary, the *order center* of each order is obtained by analyzing the information of customer orders given by the problem setting; while the *batch center* of each batch is unknown and needs to be determined. Hence, the batch centers of all batches are encoded as a part of the solution encoding in the proposed PSO algorithm.

In what follows, the solution encoding is explained in detail. In the PSO, the position vector of each particle in the search space represents a feasible solution. Consider to pick n orders. Assume that the order-picking vehicle can load at most C orders. Hence, the orders could be allocated to at most $m = \lceil n/C \rceil$ batches. As shown in Figure 2, a feasible solution is expressed by a vector of length $n + 2m$, which is divided into two parts: n numbers in the first part represent a permutation of $\{1, 2, ..., n\}$ denoted by $\langle \sigma_1, \sigma_2, ..., \sigma_n \rangle$, for each $\sigma_i \in \{1, 2, ..., n\}$, and $2m$ numbers in the second part represent the (x, y)-coordinates of m batch centers: (x_1, y_1), (x_2, y_2), ..., (x_m, y_m), where each $(x_i, y_i) \in \mathbb{Z}^2$ is a grid coordinate on the warehouse floorplan.

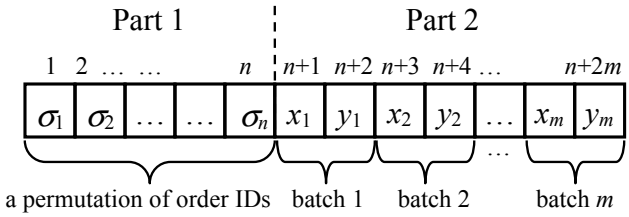

Figure 2: Illustration of the solution encoding.

3.2. Cost Evaluation (Solution Decoding)

The objective of the proposed approach is to minimize the cost of the solution, i.e., the total picker Manhattan routing distance corresponding to the solution, which is calculated in Algorithm 1.

Algorithm 1 mainly consists of two parts: order batching allocation and Manhattan routing evaluation. Given a candidate solution $(\langle \sigma_1, \sigma_2, ..., \sigma_n \rangle, (x_1, y_1), (x_2, y_2), ..., (x_m, y_m))$, the order batching allocation is determined based on content of the candidate solution. Consider each order σ_i in the left-to-right ordering of permutation $\langle \sigma_1, \sigma_2, ..., \sigma_n \rangle$ of order IDs in the first part of the solution to allocate orders to batches. Note that the order center

Algorithm 1 EVALUATE_COST()

Input: A candidate solution $(\langle \sigma_1, \sigma_2, ..., \sigma_n \rangle, (x_1, y_1), (x_2, y_2), ..., (x_m, y_m))$
Output: Cost of the candidate solution

1: **for each** $i = 1, 2, ..., n$ **do**
2: **for each** $j = 1, 2, ..., m$ **do**
3: Calculate the Euclidean distance d_{ij} between the order center of order σ_i and the batch center (x_j, y_j)
4: **end for**
5: Sort all the Euclidean distances d_{ij}'s in a non-decreasing order
6: Allocate order σ_i to the batch whose batch center has the shortest Euclidean distance to the order center of order σ_i and which does not yet achieve the maximal loading capacity of the order-picking vehicle
7: **end for**
8: **for each** $j = 1, 2, ..., m$ **do**
9: Combine the quantity of identical items in batch j
10: The picker starts at the product-collecting location
11: **while** not all items in batch j have been picked **do**
12: The picker moves from the current location to the storage location of the closest item that has not been picked
13: **end while**
14: The picker moves from the current location back to the product-collecting location
15: Compute the shortest Manhattan distance of the picker routing that cannot pass through any storage location
16: **end for**
17: Output the sum of the above m picker Manhattan routing distances

of order σ_i has been known from analyzing the problem setting as explained in the previous subsection. For $i = 1, 2, ..., n$, the Euclidean distances between the order center of σ_i and all the m batch centers in the second part of the solution are calculated (Line 3), and then are sorted in a non-decreasing ordering (Line 5). In Line 6, if order σ_i allocated to the batch with the shortest distance does not violate the maximum loading capacity of this batch, order σ_i is allocated to this batch; otherwise, it is allocated to the batch with the secondary shortest distance without violation of the maximum loading capacity until a suitable batch is found to allocate this order. Continue the similar process until all the orders have been allocated to batches.

Note that the distances between order centers and batch centers are calculated in terms of the Euclidean distance because the concepts of both centers are virtual so that it is not necessary to use the Manhattan distance that detours around the storage locations.

Also note that the idea of our order batching allocation is under the assumption that a shorter Euclidean distance between the order center of each order and the batch center of its allocated batch implies a high degree of similarity of the orders allocated to the same batch, which in turn may lead to a shorter picking route.

Next, the Manhattan routing is evaluated by a greedy strategy, and then the outputted cost is calculated. Consider each batch $j = 1, ..., m$ (Line 8). Combine the quantity of identical items in batch j (Line 9). Construct the picking route of batch j (Lines 10 – 15). Refer to the information of the shortest picker Manhattan routing distances between every two storage locations in the warehouse that can be obtained in the preprocessing procedure of the proposed PSO algorithm, which will be explained in the next subsection. Starting from the product-collecting location in the warehouse (Line 10), the picker chooses the item with the shortest picker Manhattan routing distance from the starting point as the first item to be picked, and then chooses another item with the shortest picker Manhattan routing distance from the first item as the second item to be picked. Repeat the process until all items have been picked (Line 11 – 13). Finally, the picker moves all the items back to the starting point (Line 14). Such a sequential ordering of order picking determines the picker routing Manhattan distance for batch j (Line 15). The total sum of the picker Manhattan routing distances of all batches is the total cost of the candidate solution (Line 17).

3.3. The Proposed ImPSO Algorithm

This subsection gives details of the proposed PSO approach to the joint order batching and picker Manhattan routing problem. PSO [17, 16] imitates the movement of a population of particles seeking for the optimal solution. In the k-th iteration of the PSO, each particle i moves towards the optimal solution through updating its own velocity and position on the basis of the past good experience of particle i (i.e., the best position P_k^i found so far) and the global good experience of all particles (i.e., the best position P_k^* found by all particles so far).

This work tailors an improved version of PSO (ImPSO) [14] to solve the concerned problem, in which bad experience of particles i (i.e., the worst

position B_k^i found by particle i so far) is taken into account in updating the particle velocity. In searching for an optimal solution, recording B_k^i of particle i helps prevent repetitive searches for poor positions, and hence, particle i could surpass its previous unfavorable locations and seek a better position, so that searching an optimal solution could be accelerated. That is, particle i updates its velocity and position according to good and bad experiences of particle i (P_k^i and B_k^i, respectively) and the global good experience of all particles (P_k^*). As illustrated in Figure 3, the velocity and position of particle i are updated as follows, respectively:

$$\begin{aligned} V_{k+1}^i &= w \cdot V_k^i + c_{1g} \cdot r_{1g} \cdot (P_k^i - S_k^i) \\ &\quad + c_{1b} \cdot r_{1b} \cdot (S_k^i - B_k^i) + c_2 \cdot r_2 \cdot (P_k^* - S_k^i) \qquad (9) \\ S_{k+1}^i &= S_k^i + V_{k+1}^i \qquad (10) \end{aligned}$$

where V_k^i and V_{k+1}^i are velocities of particle i in the k-th and $(k+1)$-th iterations, respectively; to avoid a too large velocity that may affect the solution search, the velocity is bounded by a constant that does not exceed length and width of the enclosing box of the warehouse floorplan; S_k^i and S_{k+1}^i are positions of particle i in the k-th and the $(k+1)$-th iterations, respectively; w is the inertia factor of the velocity; c_{1g} is the scaling factor of the particle moving towards its previous best position; c_{1b} is the scaling factor of the particle moving away from its previous worst position; c_2 is the scaling factor of the particle moving towards the global best position found so far; P_k^i (resp., B_k^i) is the best (resp., worst) position found by particle i so far in the k-th iteration; P_k^* is the previous best position found by all particles so far in the k-th iteration; r_{1g}, r_{1b}, and r_2 are random real numbers between 0 and 1.

Note that both velocity V_k^i and position S_k^i are vectors of length $n + 2m$ (see Figure 2), in which the first n values of S_k^i represent a permutation of n order IDs that is used for determining the ordering of allocating those orders into batches; the latter $2m$ values of S_k^i represent (x, y)-coordinates of m batch centers.

The proposed ImPSO algorithm is given in Algorithm 2, explained as follows. Recall that cost evaluation in Subsection 3.2 frequently adopts the shortest picker Manhattan routing distance between every two locations in the warehouse that cannot pass through any storage locations; and the distance is fixed when the warehouse floorplan is given. Hence, Line 1 of Algorithm 2 preprocesses to calculate those distances to reduce the computing

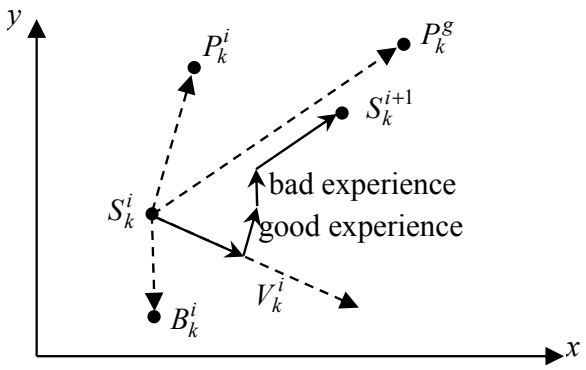

Figure 3: Illustration of the position updating equation in ImPSO.

time consumed in calculating costs. After obtaining those distances, cost of the solution represented by each particle can be calculated efficiently. The preprocessing procedure is stated as follows: for each route location, take this route location as the starting point and expand outward to compute and record the distance from this starting point to the concerned route location until all route locations of the warehouse are considered. Through this procedure, the real picker Manhattan routing distance between any two route locations can be found. For example, if the star-marked square in Figure 1 is considered as the starting point, the number attached to each white route square is the shortest picker Manhattan routing distance from the star-marked square to this square.

Since order center of each order can be obtained by analyzing the given problem setting, Line 2 of Algorithm 2 preprocess to calculate the order center coordinate of each order in advance for subsequent order batching allocation in each iteration of the main loop. Line 3 assigns initial values of position and velocity of each particle randomly on basis of the warehouse size. Line 4 calculates cost of the solution represented by each particle, i.e., the total picker Manhattan routing distance of the solution represented by this particle according to Algorithm 1. Line 5 initializes each particle's previous best position P_0^i and worst position B_0^i as the current position S_0^i, and considers cost of each particle to determine the global best position P_0^* of all particles.

After initialization, the main loop of the ImPSO optimization process is executed in Lines 6-13. In each iteration, Line 8 updates velocity of each

Algorithm 2 The Proposed ImPSO

1: Preprocess to calculate the shortest picker Manhattan routing distance between any two route locations in the warehouse
2: Preprocess to calculate the order center coordinate of each order in the warehouse floorplan
3: Initialize the velocity V_0^i and position S_0^i of each particle i randomly on basis of the warehouse size
4: Calculate cost of the solution represented by each particle according to Algorithm 1
5: For each particle i, let $P_0^i = B_0^i = S_0^i$. Calculate the global best position P_0^* found by all particles so far according to cost of each particle
6: **for each** $k = 1, 2, ..., \eta$ **do**
7: **for each** particle i **do**
8: Update velocity of particle i by Equation (9)
9: Update position of particle i by Equation (10).
10: Analyze position S_k^i of particle i to determine order batching, and calculate cost of particle i according to Algorithm 1
11: Update the best position P_k^i found by particle i so far, the worst position B_k^i found by particle i so far, and the global best position P_k^* found by all particles so far
12: **end for**
13: **end for**

particle by Equation (9). After updating velocity, Line 9 updates position of each particle by Equation (10). Line 10 analyzes the solution represented by S_k^i, where the first n values of S_k^i corresponds to the ordering of allocating orders to batches; the latter $2m$ values of S_k^i represent (x, y)-coordinates of m batch centers. Next, cost of each particle is calculated according to Algorithm 1. Line 11 updates the best position P_k^i and the worst position B_k^i found by particle i so far, and the global best position P_k^* found by all particles so far. The loop continues until the total iteration number achieves the maximal iteration number η.

3.4. Algorithm Analysis of ImPSO

Theoretical analysis of convergence and stability of the conventional PSO has been proposed in [5], and a simpler analysis method has also been proposed in [27]. The ImPSO in [14] used in this work additionally considers

previous bad experience of each particle, and hence has different analysis results. However, the previous work in [14] did not provide any theoretical analysis on algorithm convergence and stability. Therefore, this subsection refers to the analysis method in [27] to derive analysis of convergence and stability for ImPSO.

3.4.1. Convergence Analysis of ImPSO

The ImPSO additionally considers previous bad experience of each particle, so the velocity updating equation differs from the conventional PSO. Considering time division, the velocity and position updating equations are revised, respectively, as follows:

$$\begin{aligned} V_{k+1}^i &= w \cdot V_k^i + c_{1g} \cdot r_{1g} \cdot \frac{(P_k^i - S_k^i)}{\Delta t} \\ &+ c_{1b} \cdot r_{1b} \cdot \frac{(S_k^i - B_k^i)}{\Delta t} + c_2 \cdot r_2 \cdot \frac{(P_k^* - S_k^i)}{\Delta t} \qquad (11) \\ S_{k+1}^i &= S_k^i + V_{k+1}^i \cdot \Delta t \qquad (12) \end{aligned}$$

where Δt is the time step value. Substitute (11) into (12), and obtain:

$$\begin{aligned} S_{k+1}^i &= S_k^i + (wV_k^i + c_{1g} r_{1g} \frac{(P_k^i - S_k^i)}{\Delta t} \\ &\quad + c_{1b} r_{1b} \frac{(S_k^i - B_k^i)}{\Delta t} + c_2 r_2 \frac{(P_k^* - S_k^i)}{\Delta t}) \Delta t \qquad (13) \end{aligned}$$

After rearranging the above equation, obtain the following generalized equation:

$$\begin{aligned} S_{k+1}^i &= S_k^i + w \cdot V_k^i \cdot \Delta t + (c_{1g} \cdot r_{1g} - c_{1b} \cdot r_{1b} + c_2 \cdot r_2) \\ &\quad \cdot (\frac{c_{1g} \cdot r_{1g} \cdot P_k^i - c_{1b} \cdot r_{1b} \cdot B_k^i + c_2 \cdot r_2 \cdot P_k^*}{c_{1g} \cdot r_{1g} - c_{1b} \cdot r_{1b} + c_2 \cdot r_2} - S_k^i) \end{aligned}$$

The above equation can be expressed as a general gradient line-search form as follows: $S_{k+1}^i = \widehat{S}_k^i + \alpha \cdot \bar{P}_k$ where $\widehat{S}_k^i = S_k^i + w \cdot V_k^i \cdot \Delta t$, $\alpha = c_{1g} \cdot r_{1g} - c_{1b} \cdot r_{1b} + c_2 \cdot r_2$, and $\overline{P}_k = \frac{c_{1g} \cdot r_{1g} \cdot P_k^i - c_{1b} \cdot r_{1b} \cdot B_k^i + c_2 \cdot r_2 \cdot P_k^*}{c_{1g} \cdot r_{1g} - c_{1b} \cdot r_{1b} + c_2 \cdot r_2} - S_k^i$ where α can be viewed as the stochastic step size, and $\overline{P}_k$ is a stochastic search direction. The scope of the stochastic step size α is subject to the constraint of c_{1g}, c_{1b}, and c_2. If $r_{1g}, r_{1b}, r_2 \in [0, 1]$ are known, then the range of α is $[-c_{1b}, c_{1g} + c_2]$, and the

expected value is $(c_{1g} - c_{1b} + c_2)/2$. In other words, the stochastic step size is inter-dependent with c_{1g}, c_{1b}, and c_2.

Rearrange (13), and obtain:

$$\begin{aligned} S_{k+1}^i &= S_k^i(1 - c_{1g}r_{1g} + c_{1b}r_{1b} - c_2r_2) + wV_k^i\Delta t \\ &\quad + c_{1g}r_{1g}P_k^i - c_{1b}r_{1b}B_k^i + c_2r_2P_k^* \end{aligned}$$

On the other hand, rearrange (11), and obtain:

$$\begin{aligned} V_{k+1}^i &= -S_k^i \cdot \frac{c_{1g} \cdot r_{1g} - c_{1b} \cdot r_{1b} + c_2 \cdot r_2}{\Delta t} + w \cdot V_k^i \\ &\quad + c_{1g} \cdot r_{1g} \cdot \frac{P_k^i}{\Delta t} - c_{1b} \cdot r_{1b} \cdot \frac{B_k^i}{\Delta t} + c_2 \cdot r_2 \cdot \frac{P_k^*}{\Delta t} \end{aligned}$$

By the above two equations, the velocity and position of particle i in the $(k+1)$-th iteration can be presented as the following matrix form:

$$\begin{aligned} \begin{bmatrix} S_{k+1}^i \\ V_{k+1}^i \end{bmatrix} &= \begin{bmatrix} 1 - c_{1g}r_{1g} + c_{1b}r_{1b} - c_2r_2 & w\Delta t \\ -\frac{c_{1g}r_{1g} - c_{1b}r_{1b} + c_2r_2}{\Delta t} & w \end{bmatrix} \begin{bmatrix} S_k^i \\ V_k^i \end{bmatrix} \\ &\quad + \begin{bmatrix} c_{1g}r_{1g} & -c_{1b}r_{1b} & c_2r_2 \\ \frac{c_{1g}r_{1g}}{\Delta t} & \frac{-c_{1b}r_{1b}}{\Delta t} & \frac{c_2r_2}{\Delta t} \end{bmatrix} \begin{bmatrix} P_k^i \\ B_k^i \\ P_k^* \end{bmatrix} \end{aligned} \tag{14}$$

The above equation can be viewed as a discrete-dynamic system for ImPSO, where $[S^i, V^i]^T$ is constrained by an external input $[P^i, B^i, P^*]^T$.

Assuming that this dynamic system is not affected by external stimulation, and $[P^i, B^i, P^*]^T$ is a constant, the system is convergent. If letting $k \to \infty$, then $[S_{k+1}^i, V_{k+1}^i]^T = [S_k^i, V_k^i]^T$, and the matrix is transformed into:

$$\begin{aligned} \begin{bmatrix} 0 \\ 0 \end{bmatrix} &= \begin{bmatrix} -c_{1g}r_{1g} + c_{1b}r_{1b} - c_2r_2 & w\Delta t \\ -\frac{c_{1g}r_{1g} - c_{1b}r_{1b} + c_2r_2}{\Delta t} & w - 1 \end{bmatrix} \begin{bmatrix} S_k^i \\ V_k^i \end{bmatrix} \\ &\quad + \begin{bmatrix} c_{1g} \cdot r_{1g} & -c_{1b}r_{1b} & c_2r_2 \\ \frac{c_{1g}r_{1g}}{\Delta t} & \frac{-c_{1b}r_{1b}}{\Delta t} & \frac{c_2r_2}{\Delta t} \end{bmatrix} \begin{bmatrix} P_k^i \\ B_k^i \\ P_k^* \end{bmatrix} \end{aligned}$$

The above equation holds only when $V_k^i = 0$ and $S_k^i = P_k^i = B_k^i = P_k^*$. If the dynamic system is influenced by external stimulation, the solution tends to move towards the optimal condition, and, in the optimization process, the system could find a better optimal position of the particle and the global optimal position of all particles.

3.4.2. Stability Analysis of the ImPSO

The characteristic value of the dynamic matrix from (14) can be used for understanding the stable and dynamic behavior of this system. The characteristic equation of this dynamic matrix can be expressed as follows:

$$\lambda^2 - (w - c_{1g} \cdot r_{1g} + c_{1b} \cdot r_{1b} - c_2 \cdot r_2 + 1) \cdot \lambda + w = 0 \tag{15}$$

The characteristic values for the above equation are as follows:

$$\begin{aligned} \lambda_{1,2} \;=\; & (1 + w - c_{1g} r_{1g} + c_{1b} r_{1b} - c_2 r_2 \\ & \pm \sqrt{(1 + w - c_{1g} r_{1g} + c_{1b} r_{1b} - c_2 r_2)^2 - 4w} \Big) /2 \end{aligned}$$

The necessary and sufficient condition for stability of this discrete-dynamic system is $|\lambda_1| < 1$ or $|\lambda_2| < 1$. By the condition and (15), stability of ImPSO can bed inferred. The condition $\lambda_1 < 1$ implies $c_{1g} r_{1g} - c_{1b} r_{1b} + c_2 r_2 > 0$ or $c_{1b} r_{1b} < c_{1g} r_{1g} + c_2 r_2$. Since $r_{1g}, r_{1b}, r_2 \in [0, 1]$, obtain

$$c_{1b} = 0, \tag{16}$$

i.e., when $c_{1b} \to 0$, the solution of ImPSO gets more stable.

From $\lambda_1 > -1$ and $V_{k+1}^i = w \cdot V_k^i$, respectively obtain $(c_{1g} \cdot r_{1g} - c_{1b} \cdot r_{1b} + c_2 \cdot r_2)/2 - w < 1$ and $w < 1$. The two conditions let us to know that the scope of the inertia factor w of the particle velocity is:

$$\frac{c_{1g} \cdot r_{1g} - c_{1b} \cdot r_{1b} + c_2 \cdot r_2}{2} - 1 < w < 1 \tag{17}$$

From the scope, the upper bound of $c_{1g} \cdot r_{1g} - c_{1b} \cdot r_{1b} + c_2 \cdot r_2$ can be obtained as follows: $c_{1g} \cdot r_{1g} - c_{1b} \cdot r_{1b} + c_2 \cdot r_2 < 4$. Also, since $r_{1g}, r_{1b}, r_2 \in [0, 1]$, the bound implies:

$$0 < c_{1g} + c_2 < 4 \tag{18}$$

From (17), obtain:

$$\frac{c_{1g} + c_2}{2} - 1 < w < 1 \tag{19}$$

From the above, if the w, c_{1g}, and c_2 of the ImPSO meet the conditions (18) and (19), or when c_{1b} is getting smaller (inferred from the condition (16)), this system gets more stable and can converge to the equilibrium point as discussed in Subsubsection 3.4.1.

4. Implementation and Experimental Results

This section conducts a comprehensive experimental analysis on the proposed ImPSO algorithm. Initially, the experimental data and environment are explained, and a suitable parameter setting for the experimental problem is found after a number of experiments. Next, the convergence of ImPSO, the performance using different numbers of particles, and the performance using different scaling factors are analyzed.

4.1. Experimental Data and Environment

The adopted experimental data and environment are stated in detail as follows. Consider the warehouse floorplan in Figure 1, i.e., a rectangular space with 16 storage locations (squares). Since only one item is stored at each storage location, there are 16 different items. Length and width of each square are of 1 unit. This experimental problem considers 100 orders to be picked, each of which contains a random number of items ranging from 1 to 16. Each order is of small quantity and great diversity of items. The maximum capacity of the order-picking vehicle for each picking route is 4 orders, and overloading is not allowed. Since the total number of orders to be picked is 100, and the order-picking vehicle can load at most 4 orders, it takes $100/4 = 25$ batches to handle all orders. This ImPSO algorithm is implemented in C++ programming language and performs in a PC equipped with an Intel CoreTM i5 CPU@2.67GHz and memory of 6.00 GB.

4.2. The Performance under Different Scaling Factors

The velocity equation of the ImPSO (Equation (9)) includes the following three scaling factors: c_{1g} and c_{1b} for moving towards the best and the worst positions found by each particle so far, respectively; c_2 for the global best position found by all particles so far. Since different settings of the three factors would affect performance of the solution, this subsection analyzes how different combinations of these three factors influence performance, and finds a suitable combination for the subsequent experimental use.

Note that this work is the first to investigate the joint order batching and picker Manhattan routing problem. The approaches in the previous related works (e.g., [31, 18, 29]) were not designed for the picker Manhattan routing, and hence, their designs could not be fair to be used for comparison with our results. And, a key design in our proposed PSO approach is to consider each

particle's bad experience. Hence, this work compares the simulation results with and without each particle's bad experience, i.e., setting $c_{1b} = 0$ or not.

Table 1 gives the experimental results of ImPSO with different combinations of three factors (c_{1g}, c_{1b}, c_2), which include the best cost value (Best), the average cost value (Average), the worst cost value (Worst), and standard deviation of the cost values (stdDev), and the average time (Avg. time) for running 20 times of the ImPSO program. Table 1 shows that the greater the value of c_{1b} is, the worse the results become. After observing the changes in each iteration of the program, we find that the worst position of each particle causes significant damage to quality of the solution. Therefore, the experimental results of the increase of c_{1b} from 0 to 0.1 are further analyzed when c_{1g} and c_2 remain constant, as shown in Table 2. From Table 2, when c_{1b} = 0.01, the solution performance is the best on average. When c_{1b} increases from 0.01 to 0.1, all the cost values are worse than the case of $c_1b = 0.01$ but there is no trend between the value of c_{1b} and the quality of solution.

Table 1: Experimental results of ImPSO with different combinations of (c_{1g}, c_{1b}, c_2).

c_{1g}	c_{1b}	c_2	Best	Average	Worst	stdDev	Avg. time (s)
1	0	2	906	950.44	977	13.86	9.38
1	1	2	954	967.35	980	6.33	8.07
1	2	2	965	976.35	983	4.77	8.00
1	3	2	969	977.10	989	6.26	7.99
1	4	2	951	977.55	989	9.55	7.99
2	0	1	925	957.50	969	9.08	8.06
2	1	1	959	967.45	978	5.61	8.14
2	2	1	960	973.50	984	6.49	7.97
2	3	1	955	976.75	992	7.11	7.94
2	4	1	963	976.10	986	6.53	8.04

Figure 4 is the line graph plotted for the cost values of executing 100 times of the ImPSO program when $c_{1g} = 1$, $c_2 = 2$, and $c_{1b} = 1.00$, 0.10, and 0.01, respectively. From Figure 4, when $(c_{1g}, c_{1b}, c_2) = (1, 0.01, 2)$, the cost value performs better than the other two parameter settings. From the above, if the value of c_2 is greater than c_{1g}, the solution performance is better

Table 2: Experimental results of ImPSO when c_{1g} and c_2 remain constant.

c_{1g}	c_{1b}	c_2	Best	Average	Worst	stdDev	Avg. time (s)
1	0.00	2	906	950.44	977	13.86	4.69
1	0.01	2	906	948.10	975	12.74	4.57
1	0.02	2	914	951.17	976	12.66	5.35
1	0.03	2	922	954.02	984	11.11	4.57
1	0.04	2	924	954.15	976	10.39	4.95
1	0.05	2	930	954.07	973	10.77	4.75
1	0.06	2	925	956.66	976	10.44	4.69
1	0.07	2	929	956.27	975	10.20	4.62
1	0.08	2	932	957.07	975	9.93	4.63
1	0.09	2	927	958.23	980	9.84	4.63
1	0.10	2	922	958.49	979	9.32	4.64

(Table 1); a nonzero c_{1b} value could cause severe damage to the solution quality, and therefore, the c_{1b} value cannot be too larger. But if it is set at 0, the performance is worse than $c_{1b} = 0.01$ (Table 2). As a result, based on the experimental results in this subsection and the conditions derived from Subsection 3.4, the scaling factors are set at $(c_{1g}, c_{1b}, c_2) = (1, 0.01, 2)$.

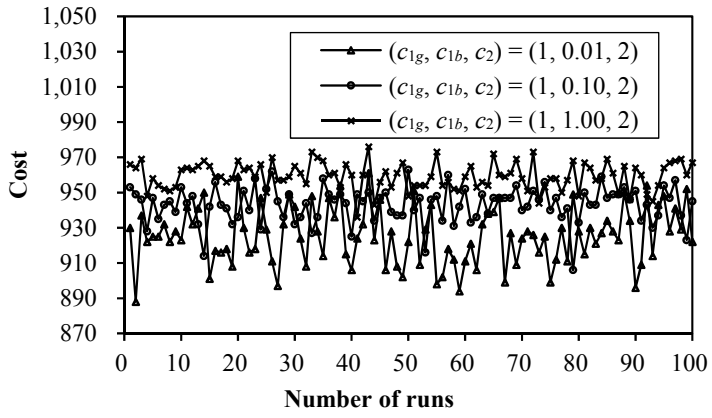

Figure 4: The cost values of 100 experiments under different c_{1b} values.

4.3. Convergence Analysis

The proposed algorithm is based on the incremental improvement on solutions, and the terminal condition of the algorithm is achievement of the maximal iteration number. The experimental goal in this subsection is to find the number of iterations that leads to convergence of the solution. For achieving this goal, considering that the number of particles set for this experiment is 180 and the scaling factor is $(c_{1g}, c_{1b}, c_2) = (1,\ 0.01,\ 2)$, we execute 5000 iterations and record their cost values in Figure 5.

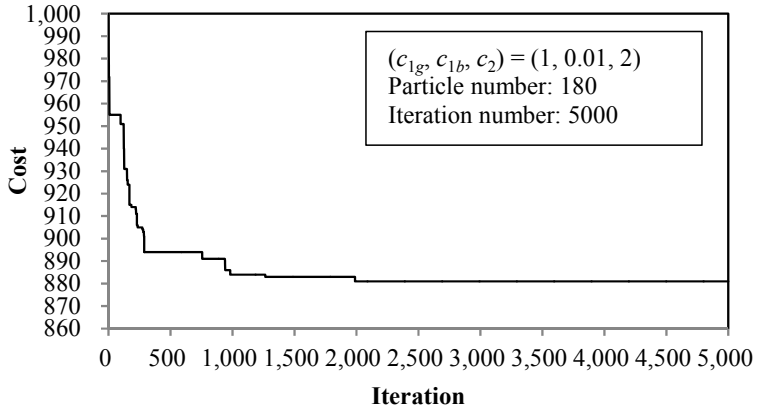

Figure 5: The plot of the cost value of each iteration.

The experimental results in Figure 5 show that the cost value rapidly falls after 300 iterations, implying an obvious improvement of solution during this period. After 300 iterations, the cost value improves slowly but keeps improving. After 2000 iterations, the cost value remains unchanged until the 5000-th iteration. Therefore, we assume that the maximal iteration number is 2000 in subsequent experiments.

4.4. The Performance under Different Particle Numbers

The spirit of the ImPSO is to quest for the incremental improvement of solutions based on previous experiences of particles. It can be anticipated that the larger the number of particles is, performance of the solution tends to be better. The experimental purpose in this subsection is to observe the relationship between the increase in number of particles and the optimization of the cost value. The scaling factors here are set at $(c_{1g}, c_{1b}, c_2) = (1, 0.01, 2)$, and the maximal iteration number is 2000. Table 3 shows the best cost value,

the average cost value, the worst cost value, and the standard deviation of the cost values, and the average time for running 100 times of the ImPSO algorithm.

Table 3: Performance of solutions under different numbers of particles.

Size	Best	Average	Worst	stdDev	Avg. time (s)
10	895	923.70	965	15.10	19.97
20	884	914.95	963	14.43	39.69
30	878	910.05	950	14.76	59.95
40	872	904.95	958	13.56	79.46
50	865	905.23	953	14.48	101.74
60	875	906.01	949	13.05	120.49
70	867	904.01	942	13.56	140.50
80	869	902.66	950	14.86	158.87
90	873	900.64	936	13.42	178.62
100	867	897.14	934	13.10	218.93
110	874	897.46	926	12.48	218.63
120	870	899.37	936	14.15	242.34
130	865	897.73	941	13.15	260.97
140	869	896.01	937	13.62	278.91
150	868	895.12	927	12.76	297.86
160	850	893.49	931	13.92	324.08
170	867	896.16	921	11.41	344.00
180	864	894.05	937	15.96	358.77

Table 3 shows that the increase in number of particles improves the average cost value. But if number of particles exceeds 40, the incremental optimization of the cost value turns slower, while the average running time keeps increasing. Therefore, it is recommended in this paper that number of particles be set at 40 to achieve a tradeoff between the solution quality and the program efficiency.

4.5. Stability Analysis

From the theoretical derivation in Subsection 3.4, when Equations (18) and (19) holds, or c_{1b} gets smaller, variance of solutions also tends to get

smaller, i.e., performance of the algorithm is more stable. The experimental purpose in this subsection is to analyze performance of the solutions using the scaling factors suggested by the conclusion of mathematical derivation in Subsection 3.4.

The scaling factors used here are set at $(c_{1g}, c_{1b}, c_2) = (1, 0.01, 2)$, which meets the conditions of Equations (18) and (19), and the inertia factor w of the particle velocity is set at 0.875 [20]. The maximal iteration number is set at 2000; and 100 independent runs of the ImPSO algorithm is executed and their best costs are recorded as plotted in Figure 6. From Figure 6, the best costs from the 100 runs of the experiment fall between 930 and 970. The experimental results show that the scaling factors satisfying the conditions of the mathematical derivation in Subsection 3.4 lead to smaller variation among the best costs in different runs of the experiment. This helps to support conclusion of the mathematical derivation.

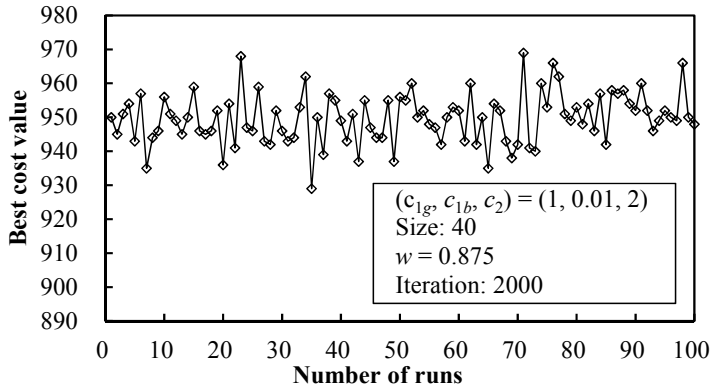

Figure 6: The best cost values of running 100 times of the ImPSO program.

5. Case Study

This section adopts the proposed ImPSO algorithm to a real-world case, and compares the experimental results with the current practice of the enterprise in the case. First, the environment and the current practice of the case are described, and then, the experimental results using the proposed ImPSO algorithm are given and compared.

Note that the previous related works (e.g., [31, 18, 29]) only provided experimental parameter settings, rather than benchmark problem instances.

In addition, those previous works can only be viewed as simplified versions of our concerned problem, i.e., without considering the picker Manhattan routing. Hence, this work evaluates performance of the proposed approach on a practical case.

5.1. Case Description and Current Practice

The personal computer (PC) market in Taiwan has a special characteristic. Aside from buying a mass-produced PC model by brand, the consumers tend to be more interested in assembling PCs at home or buying customized PCs (homebuilt or custom-built model). One of the common models is that the consumer makes a list for specifications of PCs, and requests the PC provider to assemble PCs on demand; after finishing assembly, the provider delivers the PC products to the customer. Under this model, content of components varies with different consumers' orders, and hence, each order should be customized.

This work considers some real PC assembly company that provides service of assembling and delivering customized PCs, and also sells PC parts and components by retail, i.e., customers may list specifications of the whole PC systems (motherboard, CPU, video card, memory, HDD, and power supply), and request the company to assemble the PC system. Customers may also just order some items of hardware at designated specifications without commissioning the company to assemble the whole PC system. Since the business scope of this company includes assembly of the whole PC system and retailing of PC parts and components, the company has a warehouse keeping inventory for different PC parts and components. Figure 7 shows the warehouse floorplan of this company. Since the business size and the service region of this company are small or moderate, only one picker is assigned to the duty of order picking. This picker is responsible for picking the items stated in each order daily. The company issues an order-picking vehicle to the picker. Previous experiences indicated that the picker can load items for 4 orders to the order-picking vehicle in each picking route.

The nature of the business entails great diversity of items in different orders. The required items in each order are different from order to order. Some orders may be for assembly of customized computer systems while the others are for purchase of parts and components by retail. Since there is only one picker engaged in handling orders, the company adopts the single-order-picking method to avoid complications in the order picking operation. Under this method, each order is handled individually, and has the advantage

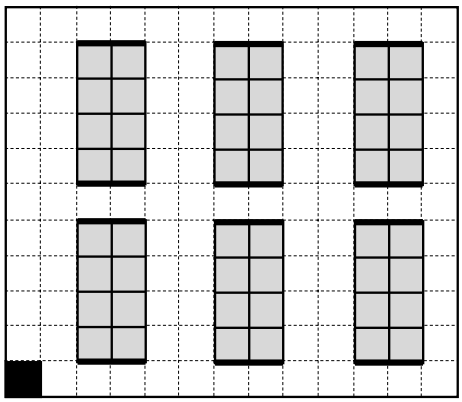

Figure 7: The warehouse floorplan of the case.

of high simplicity and flexibility in handling the orders with short lead time and little confusion caused by diversity of items in different orders. The picker can just follow requirements of orders to finish order picking one by one without being worried about misplacing items to wrong orders.

Most of the sales of this company targets individual customers, and there are hardly any large orders from other enterprises, schools, or government institutions. Since the orders may come in the form of assembly of customized computer systems or purchase of computer parts and components, the delivery date is usually a few days after the order is placed. For this company, each order is a rush order in terms of time, and the current practice of this company is on the first-come-first-served basis. That is, this company processes the orders in a chronological order. Since the picker can only pick 4 orders in each picking route for a full loading of the order-picking vehicle, the picking strategy of this company is the fixed-number batching strategy, which means that picking is performed after accumulating 4 orders.

5.2. Performance Comparison

In this case, 47 orders of this company on a particular day are processed by the proposed algorithm. Since the order-picking vehicle can handle only 4 orders in each picking route, 12 batches are needed to complete the task (where there is one batch with only 3 orders). The total picker routing distances using the ImPSO, the PSO, and the practice of this company, respectively, are compared as follows.

In the ImPSO algorithm, number of particles is set at 40; the maximal number of iterations is 2000; the three scaling factors are set at $(c_{1g}, c_{1b}, c_2) = (1, 0.01, 2)$. The main difference between the PSO and the ImPSO algorithms is consideration of the previous worst experience of each particle (i.e., $c_{1b} \neq 0$ in ImPSO but $c_{1b} = 0$ in PSO).

Table 4 shows the experimental comparison of different approaches on the best cost value, the worst cost value, the average cost value, the standard deviation of the cost values among the executed 100 runs of the experiment. From Table 4, the current practice (using the fixed-order-batching method for handling orders manually) requires 1680 units of picker routing distance in handling 47 orders. Comparatively, the total picker routing distance for handling the same orders with the proposed ImPSO approach (778 units) is less than one half of the current practice. From the results, the proposed ImPSO approach really helps shorten the total picker routing distance.

Table 4: Comparison of the results using different methods.

Method	PSO	ImPSO	Practice
Best	790	778	1680
Average	816.58	807.43	–
Worst	829	823	–
stdDev	6.703	6.853	–

In comparing PSO and ImPSO (see Table 4), the average results are very close: the average result under the PSO is 816.58 units; while the result under the ImPSO is 807.43 units. But among the 100 runs of the experiment, the best result of the ImPSO is only 778 units, which is much better than the PSO approach, and performance of the PSO in these 100 runs is also worse than the ImPSO in the best cost, average cost, and the worst cost. Therefore, the experimental result shows that introduction of previous bad experiences of particles to the PSO really improves the solution quality. Additionally, both the standard deviations of solutions under the two approach are less than 7 (accounting for about 0.8% of the cost), indicating that the solution encoding proposed in this work performs stably.

6. Conclusion

This work has designed an improved PSO (ImPSO) approach with bad experiences to solve the joint order batching and picker Manhattan routing problem under some practical constraints, in which a novel solution representation is designed for simultaneously determining the order batching allocation and the picker routing distance. In order batching allocation, similar items (in terms of the shortest picker Manhattan distance between order centers and batch centers) are classified into the same batch to be picked at one time, by which repetitive picking routes can be reduced and efficiency of picking can be improved. The ImPSO, with introduction of previous bad experiences of particles, helps to enhance efficiency of the algorithm as compared with the conventional PSO. Through each iteration of the ImPSO, the previous best and worst experiences of particles and the global previous best experience of all particles help to make particles search for the best solutions more efficiently. A comprehensive parameter analysis of the ImPSO has also been provided.

In the experimental results, a suitable parameter setting with good solution quality is found. Additionally, the experimental results show that a nonzero value of c_{1b} (the scaling factor expelling from the previous worst position of each particle) causes severe damage to the quality of the solution, but is necessary because it still helps improve the solution quality. Additionally, this solution is applicable to the problem for the orders with small quantity and great diversity of items, which are more pragmatic than the previous works. Finally, a case study is compared using the current practice, the conventional PSO, and ImPSO. The comparative results show that the proposed ImPSO performs the best, and can be effectively applied to the practical environment.

A line of future work could be focused on the hybrid with other algorithms or other advanced techniques to avoid local optimal solutions due to the rapid convergence effect of the ImPSO with bad experiences, e.g., multi-population cooperative coevolutionary algorithm [25, 28]. Additionally, the situations with different warehouse environments, multiple product-collecting locations, and multiple pickers [26] with worker learning effect [9] may also be considered. The joint order batching and picker Manhattan routing problem in this work can also be extended to the joint order batching, sequencing, and Manhattan routing problem from the previous work in [3]. Some other extensions from the vehicle routing problems could also be a line of the future

work, e.g., dynamic vehicle routing problem [24].

References

[1] T. J. Ai, V. Kachitvichyanukul, Particle swarm optimization and two solution representations for solving the capacitated vehicle routing problem, Computers & Industrial Engineering 56 (1) (2009) 380–387.

[2] T. J. Ai, V. Kachitvichyanukul, A particle swarm optimization for the vehicle routing problem with simultaneous pickup and delivery, Computers & Operations Research 36 (5) (2009) 1693–1702.

[3] T.-L. Chen, C.-Y. Cheng, Y.-Y. Chen, L.-K. Chan, An efficient hybrid algorithm for integrated order batching, sequencing and routing problem, International Journal of Production Economics 159 (2015) 158–167.

[4] C.-Y. Cheng, Y.-Y. Chen, T.-L. Chen, J. J.-W. Yoo, Using a hybrid approach based on the particle swarm optimization and ant colony optimization to solve a joint order batching and picker routing problem, International Journal of Production Economics, to appear.

[5] M. Clerc, J. Kennedy, The particle swarm-explosion, stability, and convergence in a multidimensional complex space, IEEE Transactions on Evolutionary Computation 6 (1) (2002) 58–73.

[6] R. de Koster, T. Le-Duc, K. J. Roodbergen, Design and control of warehouse order picking: A literature review, European Journal of Operational Research 182 (2) (2007) 481–501.

[7] V. Erramilli, S. J. Mason, Multiple orders per job compatible batch scheduling, IEEE Transactions on Electronics Packaging Manufacturing 29 (4) (2006) 285–296.

[8] F. P. Goksal, I. Karaoglan, F. Altiparmak, A hybrid discrete particle swarm optimization for vehicle routing problem with simultaneous pickup and delivery, Computers & Industrial Engineering 65 (1) (2013) 39–53.

[9] E. H. Grosse, C. H. Glock, The effect of worker learning on manual order picking processes, International Journal of Production Economics, to appear.

[10] S. Henn, S. Koch, G. Wäscher, Order batching in order picking warehouses: A survey of solution approaches, in: R. Manzini (ed.), Warehousing in the Global Supply Chain, Springer London, 2012.

[11] S. Henn, G. Wäscher, Tabu search heuristics for the order batching problem in manual order picking systems, European Journal of Operational Research 222 (3) (2012) 484–494.

[12] H. Hwang, M. K. Lee, Order batching algorithms for a man-on-board automated storage and retrieval system, Engineering Cost and Production Economics 13 (4) (1988) 285–294.

[13] H. S. Hwang, G. S. Cho, A performance evaluation model for order picking warehouse design, Computers & Industrial Engineering 51 (2) (2006) 335–342.

[14] A. Immanuel Selvakumar, K. Thanushkodi, A new particle swarm optimization solution to nonconvex economic dispatch problems, IEEE Transactions on Power Systems 22 (1) (2007) 42–51.

[15] V. Kachitvichyanukul, P. Sombuntham, S. Kunnapapdeelert, Two solution representations for solving multi-depot vehicle routing problem with multiple pickup and delivery requests via PSO, Computers & Industrial Engineering, to appear.

[16] J. Kennedy, The particle swarm: social adaptation of knowledge, in: Proceedings of IEEE International Conference on Evolutionary Computation, IEEE Press, 1997, pp. 303–308.

[17] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of IEEE International Conference on Neural Networks, vol. 4, IEEE Press, 1995, pp. 1942–1948.

[18] O. Kulak, Y. Sahin, M. Taner, Joint order batching and picker routing in single and multiple-cross-aisle warehouses using cluster-based tabu search algorithms, Flexible Services and Manufacturing Journal 24 (1) (2011) 52–80.

[19] R. S. Kumar, K. Kondapaneni, V. Dixit, A. Goswami, L. Thakur, M. Tiwari, Multi-objective modeling of production and pollution routing

problem with time window: A self-learning particle swarm optimization approach, Computers & Industrial Engineering, to appear.

[20] C. J. Lin, C. H. Chen, C. T. Lin, A hybrid of cooperative particle swarm optimization and cultural algorithm for neural fuzzy networks and its prediction applications, IEEE Transactions on System, Man, and Cybernetics, Part C 39 (1) (2009) 55–68.

[21] J. Luo, X. Li, M.-R. Chen, H. Liu, A novel hybrid shuffled frog leaping algorithm for vehicle routing problem with time windows, Information Sciences 316 (2015) 266–292.

[22] Y. Marinakis, G.-R. Iordanidou, M. Marinaki, Particle swarm optimization for the vehicle routing problem with stochastic demands, Applied Soft Computing 13 (14) (2013) 1693–1704.

[23] Y. Marinakis, M. Marinaki, G. Dounias, A hybrid particle swarm optimization algorithm for the vehicle routing problem, Engineering Applications of Artificial Intelligence 23 (4) (2010) 463–472.

[24] M. Mavrovouniotis, S. Yang, Ant algorithms with immigrants schemes for the dynamic vehicle routing problem, Information Sciences 294 (2015) 456–477.

[25] Y. Mei, X. Li, X. Yao, Cooperative coevolution with route distance grouping for large-scale capacitated arc routing problems, IEEE Transactions on Evolutionary Computation 18 (3) (2014) 435–449.

[26] J. C.-H. Pan, P.-H. Shih, Evaluation of the throughput of a multiple-picker order picking system with congestion consideration, Computers & Industrial Engineering 55 (2) (2008) 379–389.

[27] R. E. Perez, K. Behdinan, Particle swarm approach for structural design optimization, Computers and Structures 85 (19) (2007) 1579–1588.

[28] R. Shang, Y. Wang, J. Wang, L. Jiao, S. Wang, L. Qi, A multi-population cooperative coevolutionary algorithm for multi-objective capacitated arc routing problem, Information Sciences 277 (2014) 609–642.

[29] C. Y. Tsai, J. J. Liou, T. M. Huang, Using a multiple-ga method to solve the batch picking problem: considering travel distance and order

due time, International Journal of Production Research 46 (22) (2008) 6533–6555.

[30] G. Wäscher, Order picking: A survey of planning problems and methods, in: H. Dyckhoff, R. Lackes, J. Reese (eds.), Supply Chain Management and Reverse Logistics, Springer Berlin Heidelberg, 2004, pp. 323–347.

[31] J. Won, S. Olafsson, Joint order batching and order picking in warehouse operations, International Journal of Production Research 43 (7) (2005) 1427–1442.

[32] E. T. Yassen, M. Ayob, M. Z. A. Nazri, N. R. Sabar, Meta-harmony search algorithm for the vehicle routing problem with time windows, Information Sciences 325 (2015) 140–158.