RAIRO-Oper. Res. 58 (2024) 535–555
https://doi.org/10.1051/ro/2023156

RAIRO Operations Research
www.rairo-ro.org

PICKING SCHEDULING FOR SINGLE PICKER TO MULTI-WORKSTATIONS OF THE PART-TO-PICKER ORDER FULFILMENT SYSTEM

Jinchang Hu*, Xin Wang, Wenya Li, Chenjing Yang and Yitong Zhao

Abstract. To reduce human resource costs, the part-to-picker order fulfilment systems may have a single picker in charge of multiple workstations. And the picking speed of the picker becomes faster as the picking number increases due to the learning effect in the picking operation. In this paper, the scheduling problem to optimizing picking sequence of the picker is presented to minimize the maximum picking time, where one picker is responsible for multiple workstations. The learning effect and travel time between workstations are taken into account to improve scheduling accuracy. Two mixed integer programming (MIP) models are proposed to solve the problem, namely the rank-based model and disjunctive model. The performance of the two Mixed Integer Programming (MIP) models has been evaluated, and it has been found that they are only capable of solving small-scale problems. The rank-based model is limited to solving problems with up to 9 groups, whereas the disjunctive model can handle up to 20 groups. Therefore, the disjunctive model outperforms the rank-based model. Moreover, this paper proposes Interval Insertion NEH (IINEH) and iterative greedy (IG) algorithm to solve the large-scale problem. Numerical experiments demonstrate the effectiveness of the two methods to solve the problem, where IINEH operates faster while IG gives better results. Therefore, when faced with a large-scale problem, IINEH is recommended if a quick solution is needed. If better optimization results are needed, the decision maker can choose IG.

Mathematics Subject Classification. 90C59.

Received January 8, 2023. Accepted September 24, 2023.

1. Introduction

Today is the age of e-commerce and online shopping has become a consumption habit. The development of e-commerce has expanded people's demand for material goods and increased the requirements for quality of service. The traditional picker-to-part order fulfilment system involves the picker walking to the storage area and picking out the required items. This process requires the picker to travel excessive distances and takes up too much time, therefore this model hardly meets the requirements of modern e-commerce. Meanwhile, increased human resource costs have made the traditional picker-to-part order fulfilment system lose its cost advantage. As a result, many e-commerce companies, especially large ones, have introduced semi-automatic warehouse picking systems, *i.e.*, the part-to-picker order fulfilment systems. The inbound and outbound processes of such a system

Keywords. Part-to-picker order fulfilment system, single picker for multiple workstations, learning effect, mixed integer programming, iteration greedy algorithm.

School of Business, Shandong Normal University, Jinan 250014, P.R. China.
*Corresponding author: jinchang@sdnu.edu.cn

© The authors. Published by EDP Sciences, ROADEF, SMAI 2024

This is an Open Access article distributed under the terms of the Creative Commons Attribution License (https://creativecommons.org/licenses/by/4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

are automated and the picking of commodities from totes to orders is manual, which can significantly improve picking efficiency and reduce the number of pickers in the warehouse. In practice, there are three prevalent kinds of the part-to-picker order fulfilment systems used in e-commerce. (1) The shuttle-based storage and retrieval system, which transports totes between the storage area and workstations by shuttles, lifts and conveyors. (2) The robotic order fulfilment system, in which AGVs move the inventory pods to workstations for picking. (3) The carousel system, in which shelves are linked together and rotate in a closed-loop, and the system transports shelves with needed items to pickers. The inventory pods and shelves can be seen as consisting of multiple totes, which are loaded with the commodities needed for the order. These systems all implement the separation between the storage area and the manual picking area.

In addition, these part-to-picker order fulfilment systems are often designed based on the company's maximum throughput and the growth of the business. The sales of e-commerce companies are often related to the season and the promotions implemented by the company, such as the Double 11 in China and Black Friday in the United States. Sales tend to skyrocket at these times, and these part-to-picker order fulfilment systems are often designed in accordance with whether the system can meet this demand. In other words, the part-to-picker order fulfilment systems are often designed to achieve as much throughput capacity as possible to meet the demands of these special promotions, like setting up a sufficient number of workstations. Obviously, this capacity is redundant during the ordinary season. Therefore, when the throughput demand is not too high, the company will have one picker in charge of multiple workstations instead of assigning a dedicated picker to each workstation. When one picker is responsible for multiple picking tasks at several workstations, determining the sequence of picking tasks is a scheduling problem, which is the focus of this paper.

In reality, a batch of orders is usually classified by stock keeping unit (SKU) type before being distributed to workstations. Orders in the same class usually contain similar SKU types and are distributed to their dedicated workstations. There are several benefits to doing so. Firstly, each workstation is distributing several orders of the same class at the same time, and these orders can be picked in parallel. Here we regard these orders as a group of orders. The maximum number of orders in a group is determined by the setup of this system. Classifying orders and distributing them to dedicated workstations increases the possibility that one SKU tote out can simultaneously satisfy the needs of several orders in a group at once. This reduces the number of totes out for all SKUs in a group and ultimately improves picking efficiency. For example, if there are three orders, order A needs 2 shampoos, order B needs 3 shampoos and order C needs 1 bottle of Coke. According to the classification, we should classify orders A and B into one class and order C into another. Then we distribute orders A and B as a group to a particular workstation for simultaneous picking, so that if the totes of shampoo come out to the workstation, the picking of orders A and B can be done at the same time and the number of times the shampoo totes outbound is reduced by one. Secondly, if a workstation only needs to pick similar items of the same SKU type, the picker picking error rate is decreased. And this creates a learning effect. As a group of orders has similar SKU types, repeated picking of such orders will make the picking workers more and more familiar with the operation and thus the picking time for each picking unit will become smaller and smaller.

Obviously, with one picker responsible for multiple workstations, the picking efficiency of the part-to-picker order fulfilment system can be affected by the sequence in which workers arrive at their workstations to pick. For example, a picker may be faced with multiple workstations that are available for picking at the same time, *i.e.*, the SKU totes required for these workstations have arrived at the same time. Generally, the picker will choose the nearest workstation for picking, but this may not be the best choice for the entire picking process. In addition, learning effect makes the actual picking time subject to the picking sequence, and considering learning effect would also make the study more realistic and improve the accuracy of scheduling. Therefore, this paper investigates this picking scheduling problem influenced by learning effect to reduce picking times.

Furthermore, the following assumptions have been made in this paper. It is assumed that workstation orders are statically distributed, *i.e.*, the system can only assign the next order to a workstation once the previous order group for that workstation has been picked. It is assumed that each workstation has a sufficient buffer to hold all the totes for all SKUs of an order group. Therefore, if there is enough time, all totes for a group of required SKUs will be delivered to the workstation's buffer, which ensures picking continuity for an order

group. It is assumed that there is a large difference between the different SKU types, so we consider them to be independent and that there is no learning effect between different workstations, *i.e.*, work skills improved at one workstation can only be applied at this workstation.

The main contributions of this paper are summarized below:

(1) We studied for the first time the scenario where a single picker is responsible for multiple workstations in the part-to-picker order fulfilment system, and investigated the picking scheduling problem of the picker in this scenario. In addition, we considered the learning effect due to SKU similarity and the impact of this effect on picking operations, thus improving the accuracy of picking time calculation.
(2) We developed two types of mixed integer programming models, the rank-based model and disjunctive model, and designed experiments to measure the performance of the two models.
(3) We proposed IINEH heuristic and IG to meet the needs of different practical uses, where IINEH modified the initial order construction method and IG improved the initial solution and local search algorithm.

The paper is organized as follows. Section 2 reviews the relevant literature about order fulfilment systems, learning effects and algorithms for scheduling. Section 3 describes the considered problem, constructing two mixed integer programming (MIP) mathematical models, the rank-based model and the disjunctive model. In Section 4, we present the Interval Insertion NEH and the iterative greedy algorithm. Experimental results are provided and analyzed in Section 5. Section 6 concludes this paper and proposes further research directions.

2. Literature review

2.1. Order fulfilment system

According to the automation level, order fulfilment systems can be classified into the picker-to-part system, part-to-picker system, and completely automated picking, while completely automated picking systems are outside this paper's scope. The scheduling of picker-to-part order fulfilment systems is a traditional area of research and here we list a number of review articles that present different perspectives on this system. Cergibozan and Tasan [12] conducted a literature review of order batching operations in the framework of the order picking process and found that warehouse picking will focus on a combination of order picking, batching and routing research. Franzke *et al.* [16] investigated picker blocking with different combinations of storage allocations and picking routes, and assessed its impact. Van Gils *et al.* [41] presents a combination of tactical and operational order picking planning problems in the picker-to-part order fulfilment system and suggests guidelines on how warehouse managers can benefit from the combined planning problems. Masae *et al.* [29] identified order picker routing policies and then developed a conceptual framework for categorizing the various policies. Ahmadi Keshavarz *et al.* [1] aimed to investigate the state of the art in the adoption of the picker-to-part order fulfilment systems and carried out an extensive systematic analysis of key operational strategies, such as the simultaneous consideration of order assignment, batching, sequencing, and routing. From these review papers, we can find the picker-to-part order fulfilment system optimisation almost solves order assignment, batching, storage assignment, and routing, or combinations of these by exact algorithms, heuristic algorithms, meta-heuristic algorithms.

For the part-to-picker system, Azadeh *et al.* [4] reviewed new categories of automated and robotic handling systems, such as shuttle-based storage and retrieval systems, shuttle-based compact storage systems, and robotic mobile fulfilment systems. For each system, they categorized the literature in three groups: system analysis, design optimization, and operations planning and control. Boysen *et al.* [10] surveyed novel warehousing systems have been developed that either apply automation or implement organizational adaptions including automated picking workstations, robots, and AGV-assisted order picking systems, as well as mixed-shelves storage, dynamic order processing, and batching, zoning and sorting systems. Jaghbeer *et al.* [17] presented a systematic review and content analysis of the literature to help understanding the relevant performance aspects for automated, or partly automated order picking systems, and identifying the studied links between design and performance.

To the best of our knowledge, no previous research has considered the scenario where a single picker is responsible for multiple workstations in the picking section of the part-to-picker order fulfilment system, so research on the picking scheduling problem for this scenario does not yet exist either. For more information on this system analysis, optimal design, operational planning and control, the reader is referred to the review article Azadeh *et al.* [4] and book Yang [46]. However, on analysing the problem, we can see that it is similar to the job shop scheduling problem, with the following specific features. Firstly, one picker is responsible for several workstations in our problem, which corresponds to one operator being responsible for several machines in the job shop. Secondly, the time consumption of the picking process can be divided into two separate components, namely the required tote out time and the picker picking time, which are equivalent to the automatic production of the machine and the ready time for the worker to manually set up the raw material, respectively. However, the sequence of the two separate components of the two problems is reversed, with the job-shop scheduling problem involving manual set-up by the workers followed by automatic processing by the machine. Thirdly, the manual work of the operator is influenced by the learning curve, in our thesis if the picker repeatedly performs picking work, the picking time for one unit of items at the workstation will be reduced. Fourthly, the distance between machines should be considered. In addition, prior to picking scheduling, a batch of orders has been classified according to the similarity of SKU types in the order and divided into groups based on the number of orders that can be picked at one workstation at the same time. The system assigns a group to a dedicated workstation at a time, and the normal picking time for each group is the SKU unit picking time for that type multiplied by the number of items in that group, so the normal picking time for each group is known, much like the operator manual work time in the job shop scheduling problem. What's more, we know where all the totes required for a group's SKU are located in the storage area. Therefore, we can calculate the outbound completion time of all totes of a group, *i.e.*, the longest outbound time of all totes of the group, to ensure that all totes of the group can reach that workstation. The outbound time for a group is like the time for automatic machine processing in the job shop scheduling problem. Therefore, in the following, we mainly refer to the research of the job shop scheduling problem considering learning effect and setup time, including the model and algorithm to solve the problem.

2.2. Learning effect

The effect of learning on manufacturing productivity was first introduced by Wright [44]. Then, Biskup [7] was the pioneer of introducing learning effect into scheduling problems, proposing a learning effect model in which the processing time of a job is a function of the job's position in the sequence. Subsequently, Biskup [8] discussed the reason why the learning effect should be considered in production, and provided an overview on position-based learning effect.

As mentioned earlier, the job shop scheduling problem considering setup times is more similar to the part-to-picker order fulfilment systems studied in this paper, especially those with variable setup times. Li *et al.* [24], Anghinolfi and Paolucci [3] and Vallada and Ruiz [40] thought that the setup time depends on the sequence, *i.e.*, the setup time of the job is influenced by the work before or after the job. Wang [43] and Koulamas and Kyparisis [21] considered the past-sequence-dependent setup time $s_{[r]}$ of job $J_{[r]}$, if it is scheduled in position r, $s_{[r]} = b\sum_{i=1}^{r-1} p_{[i]}$, where $b \geq 0$ is a normalizing constant, $p_{[i]}$ is the actual processing time of a job at position i in a sequence. Pei *et al.* [34–36] present setup time s is a simple linear function of job's starting time t, that is, $s = \theta t$ where $\theta > 0$ is the deterioration rate of the setup time. Kuo [23] refered to the job independent learning model [7] and job dependent learning model [30], and proposed setup time with two kind of learning effect model, *i.e.*, $s_{i[q]} = s_i q^b$ and $s_{i[q]} = s_i q^{b_i}$, where $b \leq 0$ and $b_i \leq 0$ are the group G_i independent and dependent learning index respectively, $s_{i[q]}$ and s_i are the actual setup time and the normal setup time of group G_i respectively, and q is the group's position.

However, previous studies have hardly considered labor constraint, even in single-operator and multi-machine environments. Furthermore, if the outbound of totes needed for each workstation in this paper is analogous to setup time and the picking operation is analogous to machine processing, that setup time is known and is

independent of the picking sequences since the groups on the workstation all need to be set up before picking. The processing time, on the other hand, depends on the picking order, *i.e.*, the position of the group.

In addition, the time cost of the job consists of changing manual operation time and constant machine processing time, similar to DeJong's learning effect model. The processing time of job J_j is described by DeJong's learning curve, $p_{j,r} = p_j(M + (1-M)r^a)$, where p_j is the initial job processing time, $a \leq 0$ is the learning index, M is the incompressibility factor, r is the current position of a job in a given schedule and $1 \leq j, r \leq n$. Readers can refer to review articles of Okołowski and Gawiejnowicz [32] and Ji *et al.* [18,19] for DeJong's learning effect model. The jobs in these studies are processed sequentially on each machine one by one with DeJong's learning effect, and our scheme may result in machines waiting for operators while idle time occurs between two jobs on one machine. Therefore, to the best of our knowledge, the setting of the scheduling problem in this paper has not been studied before.

2.3. Algorithm for scheduling

Some early studies investigated scheduling problems for one operator, multi-machine systems in which operators move between machines, each with different operations, see [5,6,13]. But they all only considered one operator, two-machine flowshop or open shop models, our problem is more complex and addresses more machine of scheduling problems. In addition, we also consider the operator's travel time, because there are more machines and the operator may spend more time walking in our study, and ignoring the transport time between machines or the operator's travel time introduces larger errors.

Mixed integer programming (MIP) has been widely applied to scheduling problems, and it is often the initial approach to solving a new scheduling problem. In the context of modern solvers, Ku and Beck [22] found it valuable to understand the comparison between the various MIP models used for scheduling, and two mixed integer programming models are provided for our problem, namely rank-based formulation and disjunctive formulation, and they were first proposed by Wagner [42] and Manne [28] respectively. In addition, our problem can also be viewed as a one-operator and one-process flow shop scheduling problem with learning effect, so we consider some classical and efficient solutions to the flow shop scheduling problem.

The iterated greedy (IG) algorithm is a simple and effective type of meta-heuristics, which was first applied to solve scheduling problems by Ruiz and Stützle [38]. It starts with an initial solution generated by the NEH or improved NEH and then attempts to improve the current solution through four main phases: destruction, construction, local search, and acceptance phases. In the first stage of the iteration, some jobs are randomly removed from the current solution to obtain a partial solution. In the second phase, the removed jobs are reinserted into this partial solution in a greedy way to form a new complete solution. In the third stage, this newly generated solution is changed using a local search. Next, an acceptance criterion is used to decide whether the new solution can replace the current solution.

The NEH is a classical heuristic for flow shop scheduling problem solving proposed by Nawaz *et al.* [31], and it also serves as the initial solution of IG. This heuristic consists of two phases. First, the jobs are ordered according to an initial order, and then the first job is removed from the initial order and placed in a partial sequence that is initially empty. Next, following this initial order, each job is removed and attempted to be inserted into each possible position of the partial sequence. The position that makes the optimal objective value is chosen for the job. Most of the IG literature employs variants of the NEH procedure, Fernandez-Viagas *et al.* [15] unified the different variants of the NEH heuristic using a notation consisting of three fields.

Recently, Pan and Ruiz [33] showed that IG obtains very good results in terms of total flowtime minimization for permutation flow shop scheduling problems. Other authors have also proposed efficient IG-based algorithms to solve similar flow shop problems. Li *et al.* [24] used IG for the no-wait flow shop scheduling problem with sequence-dependent setup times and learning and forgetting effects to minimize the total flowtime. All these papers apply state-of-the-art IG methods to solve the problems related to the study of this paper, so IG methods can be used as a reference to solve our problem.

In this paper, we deal with a picker's picking scheduling problem in which the picking times of all groups depend on the group's position on its workstation, with the aim of minimizing the maximum picking time for

all orders. We provide two mixed integer programming models for this scheduling problem and propose an improved iterative greedy algorithm for larger size problems.

3. Problem definition and MIP models

Our picking scheduling problem, where one picker is responsible for multiple workstations, can be formulated as follows. There are M workstations in a part-to-picker order fulfilment system, all picked by one picker. As described above, we classify orders according to SKU types and distribute the same class of orders to the same workstation, where each order contains only one type of SKU. The orders of a workstation are divided into multiple groups, and assume that workstation i has n_i groups. All workstations pick a total of N groups, $N = \sum_{i=1}^{M} n_i$. Let n_{ig} be the commodities number of the gth group of workstation i and u_i be the unit commodity picking time of workstation i. The system delivers the required totes from the storage area to the workstations, the pickers pick the commodities into the order bins, and then the system transports the totes back to the storage area and repeats the series until all orders are completed. Symbols s_{ig} and p_{ig} denote the normal picking time and outbound time of the gth group of workstation i, where $s_{ig} = n_{ig}u_i$, and assume that $p_{i(n_i+1)} = 0$. After the picker finishes picking at a workstation, he or she selects an alternative workstation that is available for picking, the picker's travel time between workstations i and l is t_{il}, here $t_{il} = t_{li}$. We set the outbound time to be sufficient to deliver the required totes for a group of orders from the storage area to the workstations, so there is no preemption of orders from different workstations, *i.e.*, once a picker arrives at a workstation, he or she must complete the group of orders. However, if the required totes have not yet arrived, the picker needs to wait at the workstation where the pickers are scheduled to reach. We consider the group-position and workstation-based learning effect for the picking operation. The actual picking time of the gth group of workstation i is $s'_{ig} = s_{ig}g^a$, where $a \leq 0$ is the learning index. The objective is to minimize the maximum picking time for all orders, $C_{\max}$.

3.1. Rank-based model

We provide a MIP mathematical model for our one picker picking scheduling problem with group-position and workstation-based learning effect for picking time. The other parameters and variables in the model are described below and then the proposed model is given.

Decision variables: z_{kig} (binary variable), if group k is scheduled at the gth position on workstation i to be processed, $z_{kig} = 1$ otherwise $z_{kig} = 0, k = 1, 2, \ldots, N; i = 1, 2, \ldots, M; g = 1, 2, \ldots, n_i$.

Auxiliary variables: x_{ki} (continuous variable), the earliest time the picker can be scheduled to pick at the workstation i after the kth group picking is finished, $k = 1, 2, \ldots, N; i = 1, 2, \ldots, M$. In more detail, x_{ki} is an auxiliary time record that refers to the earliest time when the next group can be picked after picking group k (assuming group k belongs to the gth group on workstation i). So, it means that the picker needs to have picked group k on workstation i and a group after group k (not necessarily $k+1$, but certainly $g+1$, which represents the group after the gth group on workstation i) has already left the warehouse and arrived at workstation i. So, x_{ki} needs to add the picking time and the next group outbound time of that workstation after starting picking.

Objective function:

$$\min C_{\max}. \tag{1}$$

Constraints:

$$\sum_{i=1}^{M}\sum_{g=1}^{n_i} z_{kig} = 1, \qquad k = 1, 2, \ldots, N \tag{2}$$

$$\sum_{k=1}^{N}\sum_{g=1}^{n_i} z_{kig} = n_i, \qquad i = 1, 2, \ldots, M \tag{3}$$

$$\sum_{k=1}^{N} z_{kig} = 1, \qquad i = 1,2,\ldots,M;\quad g = 1,2,\ldots,n_i \tag{4}$$

$$x_{ki} \geq p_{i1}, \qquad i = 1,2,\ldots,M;\quad k = 1,\ldots,N \tag{5}$$

$$x_{ki} \geq x_{(k-1)i} + \sum_{g=1}^{n_i} z_{kig}\left(s'_{ig} + p_{i(g+1)}\right), \qquad i = 1,2,\ldots,M;\quad k = 1,2,\ldots,N \tag{6}$$

$$\begin{aligned} x_{ki} \geq\ & x_{(k-1)l} - \sum_{h=1}^{n_l} z_{(k-1)lh} p_{l(h+1)} \\ & + \sum_{l=1}^{M}\sum_{g=1}^{n_i}\sum_{h=1}^{n_l} z_{kig} z_{(k-1)lh} t_{il} \\ & + \sum_{g=1}^{n_i} z_{kig}\left(s'_{ig} + p_{i(g+1)}\right) \\ & - V\left(1 - \sum_{l=1}^{M}\sum_{g=1}^{n_i}\sum_{h=1}^{n_l} z_{kig} z_{(k-1)lh}\right), \end{aligned} \qquad \begin{aligned} & i = 1,2,\ldots,M;\quad l = 1,2,\ldots,M;\quad i \neq l;\quad k = 1,2,\ldots,N \end{aligned} \tag{7}$$

$$C_{\max} \geq x_{Ni}, \qquad i = 1,2,\ldots,M \tag{8}$$

$$z_{kig} \in \{1,0\}, \qquad i = 1,2,\ldots,M;\quad k = 1,3,\ldots,N;\quad g = 1,2,\ldots,n_i. \tag{9}$$

The objective is to minimize the maximum picking time. Constraint (2) ensures every group is picked once. Constraint (3) ensures n_i groups are picked on workstation i. Constraint (4) specifies that only one group be scheduled at the every position in all workstations. Constraint (5) enforces that all groups can only be picked after all totes of that group have been reached. The constraint (6) denotes x_{ki} should add the picking time of group k and the outbound time of the group after the gth group on workstation i, when group k belongs to the gth group on workstation i, that is, both k and $k-1$ are on workstation i. Note: regardless of whether group k belongs to the gth group on workstation i, it is certain that $x_{ki} \geq x_{(k-1)i}$. The constraint (7) is another case, when k and $k-1$ are not on the same workstation and k belongs to the gth group on workstation i, while $k-1$ belongs to the hth group on workstation l. In this case x_{ki} is not only limited by constraint (6), but also by the time of arrival of the picker at workstation i from workstation l. The time that the picker can depart from workstation l is $x_{(k-1)l} - \sum_{h=1}^{n_l} z_{(k-1)lh} p_{l(h+1)}$ (according to the definition of x_{ki}, we need to subtract the outbound time of the $(h+1)$th group on workstation l). The time for the picker to walk from workstation l to workstation i is $\sum_{l=1}^{M}\sum_{g=1}^{n_i}\sum_{h=1}^{n_l} z_{kig} z_{(k-1)lh} t_{il}$, in addition, according to the definition of x_{ki}, we also need to add the picking time of group k and the outbound time of the group after the gth group on workstation i. Finally the product of V is used so that constraint (7) holds only when k belongs to the gth group on workstation i and $k-1$ belongs to the hth group on workstation l. In our model, we assign $V = \sum_{i=1}^{M}\sum_{g=1}^{n_i}(s_{ig} + p_{ig}) + (N-1)\cdot \max_{i,l=1,2,\ldots,M} t_{il}$, since none of the workstations can take more time than the sum of the normal picking time and outbound time for all groups, plus the sum of the pickers' travel time, here we use the largest value for all the time between any two machines. We use the same V value in the disjunctive model. Constraint (8) defines the maximum picking time. Finally, constraint (9) defines z_{kig} to be a binary variable.

This Rank-based model includes $z_{kig} z_{(k-1)lh}$, which makes constraint (7) nonlinear inequalities. But it is clear that $z_{kig} z_{(k-1)lh}$ and $(z_{kig} + z_{(k-1)lh} - 1 + |z_{kig} + z_{(k-1)lh} - 1|)/2$ are equivalent. In addition, for constraint containing absolute value operations, such as $x > |y|$, we can modify the constraint to a combination of $x > y$ and $x > -y$, they are both linear constraints. So, we can also regard the rank-based model as an MIP model.

3.2. Disjunctive model

We sort all groups by workstation index, *i.e.*, group $1, 2, \ldots, N$ belongs to workstation $1, 2, \ldots, M$ successively. Let w_k represents the kth group's workstation, s'_k and p'_k represent the actual picking time and the outbound time of the kth group. Assume the kth group is the n'_kth group of workstation w_k.

To build the relationship of groups with the index range of group of the workstation, we define ρ_k and σ_k as minimum and maximum group index of kth group's workstation. Then, to establish the relationship between the group and the range of group indexes of the workstations in that group, we define ρ_k and σ_k as the minimum and maximum group index of the kth group's workstation. For example, when the 1st group belongs to the workstation 1 and the 2nd group to the $(n_1 + n_2)$th group all belong to the workstation 2, then $\rho_1 = 1, \sigma_1 = 1, \rho_2 = 2$ and $\sigma_2 = n_1 + n_2$. Meanwhile, let τ_{kj} be the travel time of the picker between the workstations of the kth and the jth group. We give the methods to calculate $\rho_k, \sigma_k, s'_k, p'_k$ and $\tau_{kj} (k, j = 1, 2, \ldots, N)$ as follows, and they can be regarded as known parameters:

$$\rho_k = \sum_{i=1}^{w_k - 1} n_i + 1 \tag{10}$$

$$\sigma_k = \sum_{i=1}^{w_k} n_i \tag{11}$$

$$n'_k = k - \sum_{i=1}^{w_k - 1} n_i \tag{12}$$

$$s'_k = s_{w_k n'_k} (n'_k)^a \tag{13}$$

$$p'_k = p_{w_k n'_k} \tag{14}$$

$$\tau_{kj} = t_{w_k, w_j}. \tag{15}$$

Besides, we define decision variables.

Decision variables: z_{jk} (binary variable), if the jth group precedes the kth group in picking, $z_{jk} = 1$, otherwise $z_{jk} = 0$, $k, j = 1, 2, \ldots, N$.

Auxiliary variables: x_k (continuous variable), the start time of picking the kth group, $k = 1, 2, \ldots, N$

$$\min C_{\max}. \tag{16}$$

Constraints:

$$x_k \geq p'_k, \qquad k = 1, 2, \ldots, N \tag{17}$$

$$x_j \geq x_k + s'_k + p'_j - V \cdot z_{jk}, \qquad j = 1, 2, \ldots, N, \quad \rho_j \leq k \leq \sigma_j, \quad k > j \tag{18}$$

$$x_k \geq x_j + s'_j + p'_k - V \cdot (1 - z_{jk}), \qquad j = 1, 2, \ldots, N, \quad \rho_j \leq k \leq \sigma_j, \quad k > j \tag{19}$$

$$x_j \geq x_k + s'_k + \tau_{kj} - V \cdot z_{jk}, \qquad j = 1, 2, \ldots, N, \quad k > \sigma_j \text{ or } k < \rho_j, \quad k > j \tag{20}$$

$$x_k \geq x_j + s'_j + \tau_{kj} - V \cdot (1 - z_{jk}), \qquad j = 1, 2, \ldots, N, \quad k > \sigma_j \text{ or } k < \rho_j, \quad k > j \tag{21}$$

$$C_{\max} \geq x_k + s'_k, \qquad k = 1, 2, \ldots, N \tag{22}$$

$$z_{jk} \in \{0, 1\}, \qquad k, j = 1, 2, \ldots, N. \tag{23}$$

The objective function is stated in (16). Constraint (17) ensures each group starts picking after the arrival time of all its totes. The disjunctive constraint sets (18)–(21) ensure that no two groups can be picked on the same workstation at the same time. If the two groups are on the same workstation, the latter group must start being picked after the former group has finished, *i.e.*, constraints (18) and (19), else the travel time of the picker between the two workstations must be considered, *i.e.*, constraints (20) and (21). Symbol V has to be assigned as a large enough value to ensure the correctness of these constraints. Constraint (22) ensures that the

TABLE 1. Comparison of two MIP models.

Model	Number of binary variables	Number of constraints	Continuous variables
Rank-Based	N^2	$M^2N + MN + 2M + 2N$	NM
Disjunctive	N^2	$N^2 + N$	N

maximum picking time is after the completion time of the last group. Finally, constraint (23) defines z_{jk} to be a binary variable.

In summary, this problem is a picker's picking scheduling problem and also considers the distance between two workstations. Therefore, the start of picking for each group must be later than the arrival of the picker, which is constraint (7) in the rank-based model and constraint (20), (21) in the disjunctive model. These all are different from the traditional job shop scheduling problem model, which is reviewed in Ku and Beck [22]. Table 1 shows the number of binary variables, constraints and continuous variables of two MIP models. Because both models are difficult to solve and can only solve small-scale instances, the next sections propose heuristic procedures for solving large problems.

4. ITERATED GREEDY ALGORITHM

The iterated greedy (IG) algorithm is a simple and effective type of meta-heuristics, we adopt this IG framework, which is shown in Algorithm 1. The main improvement of the IG framework is made to its initial solution. Classical IG usually uses NEH as the initial solution, while our NEH modifies the first field initial order construction method, calling it Interval Insertion NEH (IINEH).

Reference literature by Fernandez-Viagas *et al.* [15], NEH starts by sorting all the orders to form the initial order, which mainly includes the following sorting criteria. (1) rand: Jobs are randomly ordered. This order is used by Ribas *et al.* [37]. (2) SD: Non decreasing sum of processing times (original order of the NEH) of the jobs. This order is used by Kalczynski and Kamburowski [20] and Nawaz *et al.* [31]. (3) AD: sum of the mean and deviation of the processing times This order is used by Dong *et al.* [14]. Our proposed IINEH is new sorting criteria, which allows two adjacent groups of the same workstation to be inserted in the partial sequence with the largest possible time interval so that they are as far apart as possible in the partial sequence and reduce the picker's waiting time. For this purpose, we define the group interval Int_i for the ith workstation and obtain the weight w_k^i for the kth group of the ith workstation, which are calculated as $Int_i = N/n_i$ and $w_k^i = k \cdot Int_i$, respectively. Then, all groups are sorted in ascending order according to their weights to generate an initial order (π'). Next, as in the traditional NEH rule, the first group in the initial order (π'_1) is removed and inserted into each position of the partial sequence to obtain the different insertion results. The insert position that minimizes the maximum picking time of the new partial sequence is chosen and this group is inserted. This operation is repeated until all groups in the initial order are removed to obtain the complete group sequence π. The pseudocode of the IG algorithm is presented in Algorithm 2.

To further improve the search capability, a local search algorithm is added to the algorithm. The operation of our local search is to swap the positions of two fragments in the group sequence, the fragments consist of the parameters d_1 consecutive groups. The operation is executed L_1 times per iteration, and after the execution, we make a judgment on the new group sequence. If the maximum picking time of the new group sequence obtained by the above two stages is shorter than the original one, the new group sequence is accepted, otherwise the original group sequence remains unchanged. The pseudocode of the proposed Local search procedure is given in Algorithm 3.

The local search process enhances the reinforcement of the search algorithm. However, Blum and Roli [9] indicated the balance between intensification and diversification is crucial to avoid getting trapped in a local

Algorithm 1. Iterated greedy (IG).

Input: L_1, d_1, L_2, d_2;
Output: π;
$\pi_0 \leftarrow$ Interval Insertion NEH;
while Time condition is not satisfied **do**
    $\pi \leftarrow$ local search (π_0);
    $\pi \leftarrow$ Destruction & construction (π);
    $\pi_0 \leftarrow \pi$
end while

Algorithm 2. Interval Insertion NEH.

Output: π;
for $i = 1$; $i \leq M$; $i++$ **do**
    Calculate the group interval of workstation i: $Int_i = N/n_i$;
    Calculate the weight of the kth group on workstation i: $w_k^i = k \cdot Int_i, k = 1, 2, \ldots, n_i$;
    $\pi' \leftarrow$ Sort all groups in non-descending order according to their w_k^i;
end for
$\pi \leftarrow \pi'_1$; // Move the first group of π' into π
for $i = 2$; $i \leq N$; $i++$ **do**
    $\pi \leftarrow$ best permutation obtained by inserting π'_1 in any possible positions of π;
    Removed π'_1 from π';
end for
return π;

Algorithm 3. Local search.

Input: π, L_1, d_1;
Output: π;
for $i = 1$; $i \leq L_1$; $i++$ **do**
    $k_1, k_2 \leftarrow Random(1, N - d_1)$; // $Random(x, y)$ means generate a random integer between x and y
    $\pi' \leftarrow \pi$;
    for $j = 0$; $j < d_1$; $j++$ **do**
        Exchange $\pi(k_1 + j)$ and $\pi(k_2 + j)$ of π';
    end for
    if $C_{\max}(\pi') < C_{\max}(\pi)$ **then**
        $\pi \leftarrow \pi'$;
        $C_{\max}(\pi) \leftarrow C_{\max}(\pi')$;
    end if
end for
return π;

optimum. The destruction and reconstruction process is performed on the sequence to increase the diversity of the search process, which significantly changes the original sequence and increases the possibility of the algorithm jumping out of the local optimal solution. The algorithm iterates L_2 times, and each iteration consists of three phases: Destruction, Reconstruction and Acceptance Criteria. (1) Destruction: d_2 groups are randomly removed from the original sequence and put into a temporary groups sequence in turn. (2) Reconstruction: similar to the NEH rule, remove the first group in the temporary group sequence, insert it into each position of the original group sequence, and choose the position that minimizes the maximum picking time of this group sequence for insertion, and repeat the above operations until all groups in the temporary group sequence are removed and the original group is restored to a new complete group sequence. (3) Acceptance Criterion: if the maximum picking time of the new group sequence obtained by the above two stages is shorter than the original one, the new group sequence is accepted, otherwise the original group sequence remains unchanged. This algorithm can

finally obtain an updated group sequence. Destruction & Reconstruction algorithm is formally described in Algorithm 4.

Algorithm 4. Destruction & Reconstruction.

Input: π, L_2, d_2;
Output: π;
for $i = 1$; $i \leq L_2$; $i++$ **do**
  $\pi' \leftarrow \pi$;
  for $j = 0$; $j < d_2$; $j++$ **do**
    Remove one group at random from π' and insert it into π'';
  end for
  for $j = 0$; $j < d_2$; $j++$ **do**
    $\pi' \leftarrow$ best permutation obtained by inserting π_1'' in any possible positions of π';
  end for
  if $C_{\max}(\pi') < C_{\max}(\pi)$ **then**
    $\pi \leftarrow \pi'$;
    $C_{\max}(\pi) \leftarrow C_{\max}(\pi')$;
  end if
end for
return π;

5. Computational and statistical experimentation

Computational experiments are conducted to evaluate the performance of two MIP models and IG algorithm. Five factors are considered in our experiments, including the total number of groups (N), learning effect index (a), unit picking time (u, the units are seconds), distances between adjacent workstations (d, the units are in meters) and workstation number (M). To determine the total number of groups N and facilitate statistical categorization, we define X, which denotes the average number of groups per workstation, such that $N = XM$, hence X and M together determine the size of the problem in the experiment. N groups are randomly assigned to each workstation, ensuring that at least one group is picked at each workstation. The items number of every group in all instances also supports a uniform distribution $U[6, 20]$, and the normal picking time for each group is its items number multiplied by the unit picking time for that SKU type. We consider that all workstations are set up on the same side of the storage area as one column and that the spacing between any two adjacent workstations is the same, so we only describe the distance between two adjacent workstations as follows. In this study, all experimental tests are conducted on a personal computer with Intel(R) Core(TM) i7-8550U CPU @ 1.80 GHz and 16 GB RAM. We use the Gurobi optimizer 8.0 for the rank-based model and the disjunctive model, the computational time of Gurobi is limited to 3600 s. The gap between the lower bound and the tardiness of the best feasible solution is recorded if Gurobi reaches the time limit. The computational experiments are divided into the following parts according to different purposes: (1) comparison of the disjunctive model and the rank-based model; (2) comparison of the disjunctive model and the proposed IG; (3) parameter calibration of the proposed IG methods; (4) performance of the proposed IG and analysis of the influencing factors; (5)performance of the proposed IINEH; (6) comparison IINEH and IG with the traditional algorithms.

We use the following simulation method as a comparison algorithm for the MIP model and the proposed IG algorithm. The basic idea of the simulation is to enable the picker to pick as early as possible, regardless of the workbench, in order to reduce the picker's idle time. If there are multiple workstations that can be picked, it is logical that the picker should reach the closest workstation in order to be able to pick as early as possible. This idea is generally consistent with the actual picking process. To describe the simulation method, we have the following definitions. Let the time for the picker to finish picking the current workstation group be t_c with an initial value of 0. Let the time for each workstation to finish picking the current group be $t_i^w, i = 1, 2, \ldots, M$

with an initial value of 0 for every i. The workstation where the picker is currently located is e with a random initial value. The symbol n_i^c represents the number of groups completed at workstation i with an initial value of 0. The definitions of t_{il}, s_{ig} and p_{ig} are the same as in Section 3. A formal description of the simulation is given in the Algorithm 5.

Algorithm 5. Simulation.

Output: π;
1: $e \leftarrow$ a random workstation; $\pi \leftarrow \emptyset$
2: **for** $i = 1; i \leq N; i++$ **do**
3: $S \leftarrow$ the workstations set with groups to pick; $\mathbb{K} \leftarrow \emptyset$;
4: **for** each $i' \in S$ **do**
5: $\mathbb{K} \leftarrow \mathbb{K} \cup \max\{t_c + t_{i',e}, t_i^w + p_{i',n_i^c+1}\}$;
6: **end for**
7: $t_c'' \leftarrow \min\{\mathbb{K}\}$; $e \leftarrow$ the workstation i' meet $t_c'' = \min\{\mathbb{K}\}$; $t_c, t_e^w \leftarrow t_c'' + s_{e,n_i^c+1}$; $n_i^c \leftarrow n_i^c + 1$; $\pi \leftarrow \pi \cup e$;
8: **end for**
9: **return** π;

In addition, to compare two algorithms or methods, and to measure the quality of their obtained solutions, we present the relative percentage deviation, RPD_A^B, which is computed for each experiment according to the following equation:

$$\mathrm{RPD}_A^B(\%) = 100 \times \frac{C_A - C_B}{C_B} \tag{24}$$

where C_A and C_B denote the objective value of algorithms or methods A and B for every instance, respectively.

5.1. Comparision between the disjunctive model and rank-based model

In this part, we compare the rank-based model with the disjunctive model. We test two workstations with the number $M = 3$ and 4, three X, 2, 3 and 4. The learning effect index $a = -0.15$ and the distance d between adjacent workstations is 6. Unit picking time and outbound time of every workstation are generated from a uniform distribution $U[5, 10]$ and $U[60, 300]$, respectively. We record CPU time and gap of the disjunctive and rank-based model of every instance, and we use the relative percentage deviation RPD_{ran}^{dis} to compare the disjunctive model and rank-based model, where *ran* and *dis* denote the rank-based and disjunctive model, respectively.

From Table 1 we can see that the constraints and continuous variables number of the rank-based model is significantly larger than that of the disjunctive model, so the rank-based model needs more running time to achieve optimality. Table 2 shows a comparison between the rank-based model and the disjunctive model. The rank-based model can only complete solving problems smaller than $M = 3, X = 3$ and $M = 4, X = 2$ in one hour, while the disjunctive model can solve problems of this size quickly. As shown in Table 2, the value of RPD_{ran}^{dis} is small for all scales, indicating that the final results of the two models are not very different. However, the mean and maximum gaps of the rank-based model are large, especially for the larger size cases, which indicates that the lower bounds of the solutions of the rank-based model in Gurobi are very small. In addition, problems with more than 4 workstations and 8 groups take too much time to solve, so we think the rank-based model is not suitable to solve our problem. Therefore, we just analyze the performance of the disjunctive model and IG as shown below.

5.2. Calibration of the proposed IG methods

There are four parameters that can affect the performance of the proposed IG algorithm: L_1, d_1 in the local search algorithm, and L_2, d_2 in Destruction & construction algorithm. L_1 and L_2 are tested at 3, 5 and 7 levels respectively, d_1 and d_2 are tested at 2, 3, 4 and 5 levels respectively. Therefore, a total of $3 \times 3 \times 4 \times 4 = 144$

TABLE 2. Comparison between the disjunctive model and rank-based model.

M	X	RPD_{ran}^{dis}		Disjunctive model CPU time		Rank-based model CPU time		Rank-based model gap (%)	
		Mean	Max	Mean	Max	Mean	Max	Mean	Max
3	2	0	0	0.02	0.02	0.67	1.25	0	0
	3	0	0	0.07	0.10	40.21	92.99	0	0
	4	0.12	0.62	0.29	0.57	824.124	3600	22.62	90.49
4	2	0	0	0.10	0.13	7.67	10.93	0	0
	3	0.38	1.89	0.69	1.25	2894.41	3600	66.89	93.74
	4	0.19	0.46	9.59	19.27	2886.04	3600	94.93	95.77

TABLE 3. The ANOVA results of the parameters of the proposed IG.

Source	Sum of squares	Df	Mean square	F-ratio	p-value
Main effects					
L_1	3.88×10^{-6}	2	1.94×10^{-6}	0.06	0.95
L_2	9.06×10^{-4}	2	4.53×10^{-4}	13.06	2.16×10^{-6}
d_1	1.39×10^{-3}	3	4.65×10^{-4}	13.36	1.05×10^{-8}
d_2	5.44×10^{-4}	3	1.81×10^{-4}	5.23	1.33×10^{-3}
Residual	0.44	12 949	3.47×10^{-5}		
Total (corrected)	0.45	12 959			

different configurations are generated. The experimental scenario is determined by the number of workstations M and the multiplier X, where M takes three values 6, 8 and 10, and X takes three values 8, 9 and 10, so there are $3 \times 3 = 9$ experimental scenarios. Ten samples are tested under different scenarios and for different combinations of parameter configurations. Therefore, there are $144 \times 9 \times 10 = 12\,960$ test results. In each test, the learning effect a is -0.15, the distance between adjacent workstations d is 6, unit picking time and outbound time of every workstation are generated from a uniform distribution, $U[5, 10]$ and $U[60, 300]$, respectively. To make the running time of the algorithm proportional to the complexity of the scenario, the time limit for each test is set to $0.5MX$ seconds. Once the tests are performed, we calculate the effectiveness of the algorithm for an instance by RPD_s^{IG}, where s and IG denote simulation and IG, respectively.

The influence of the parameters on the experiment is analyzed by the Analysis of Variance (ANOVA) technique. The three main assumptions of ANOVA, namely normality, homogeneity of variance (or homoscedasticity) and independence of residuals, can be verified. Table 3 shows that the F-values of parameters L_2, d_1 and d_2 are large, and the p-values are all much less than the significance level of 0.05. It indicates that these three parameters can have a significant impact on the experiment, and their different values can bring significant differences to the experimental results. However, the p value of parameter L_1 is greater than the significance level of 0.05, so it has no significant effect on the experiment.

Figure 1 shows the relevant means plots with 95% Least Significant Difference (LSD) confidence intervals for the proposed IG algorithm. As with the analysis above, the confidence interval of parameter L_1 at each level is not significantly different, the intervals of all candidate values are overlapping, but it can get a slightly better result when the value is 7. It can also be seen that it will obviously have better results when parameters L_2 and d_2 are both taken as 3. And it seems that $d_1 = 4$ provides better results among all candidate levels, although it is not statistically significantly different from that when taken as 5.

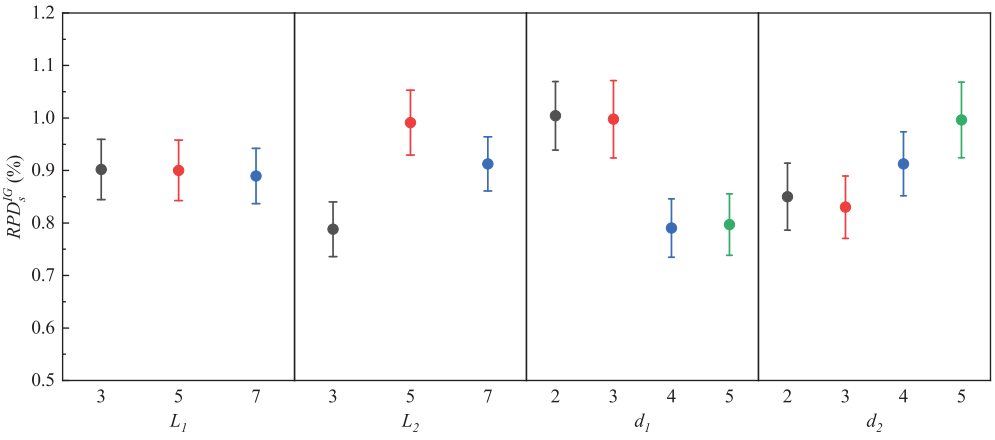

FIGURE 1. Means plots and 95% confidence level LSD intervals for all the factors.

5.3. Comparison the disjunctive model with IG

In this part of the experiments, we compare the performance of the disjunctive model and IG algorithm over small-sized problems. We test three workstation numbers $M = 3$, 4 and 5, and four X, 3, 4, 5, and 6. For learning effect indexes a, we test -0.51, -0.32, -0.15, and 0, corresponding to four different values of the learning effect, 70%, 80%, 90%, and 100%. It means that the actual processing time of the job is 70%, 80%, 90%, and 100% of the previous one respectively. Wu *et al.* [45] and Yin *et al.* [47,48] have used the same learning effect indexes value assignment method. For more detailed calculations, please refer to the literature by Biskup [8]. Unit picking time and outbound time of every workstation are generated from a uniform distribution $U[5, 10]$ and $U[60, 300]$. Thus, this part of the experiment has a total of $3 \times 4 \times 4 = 48$ experimental scenarios, each containing 10 random instances. We use the disjunctive model and the IG algorithm to solve each instance and perform simulations using Algorithm 5. The termination condition of IG is set to a time elapsed of $0.5MX$ seconds. Another relative percentage deviation RPD_s^{dis} is employed to evaluate the quality of solutions of the disjunctive model.

We summarize the results in Table 4, which includes the mean and maximum CPU time, the mean and maximum gaps of the disjunctive model, $\mathrm{RPD}_s^{\mathrm{IG}}$ and mean CPU time of IG. In order to express more clearly the difference in the optimization effect of the two methods, we list $\mathrm{RPD}_s^{dis} - \mathrm{RPD}_s^{\mathrm{IG}}$ instead of RPD_s^{dis}.

Table 4 shows that the disjunctive model can solve these problems with up to 20 groups. Compared to IG, the disjunctive model outperforms IG when the problem size is small, *i.e.*, less than 5 workstations and 25 groups, although there are some scenarios that the disjunctive model cannot solve in less than one hour. However, when the problem is larger than this size, IG gives better results. In summary, for small size problems, the disjunctive model and IG have similar optimization results, since the value of disjunctive-IG is mostly 0 or close to 0. The two methods can improve the picking efficiency by about 2.42% on average. Moreover, when $M \leq 4$ and $X \leq 4$, the disjunctive model is faster than IG, which has a significant advantage especially for the extremely small size problems. However, when the sizes are larger than that, the CPU time of the disjunctive model increases considerably and the instability is enhanced, as shown by the different mean and maximum values of the CPU time of the disjunctive model. In conclusion, the disjunctive model is more suitable for solving problems with less than 4 workstations and 16 groups. For larger problems, IG is a better choice.

5.4. Performance of the proposed IG at different system parameters

In this part of the experiments, the proposed IG algorithm is tested with more scenarios. The learning effect a used for the experiments takes the same values as in Section 5.3. To further analyze the effect of different

TABLE 4. Comparison between the disjunctive model and IG under different learning effect.

M	X	a	Disjunctive model						IG	
			gap (%)		CPU time		$RPD_s^{dis} - RPD_s^{IG}$		RPD_s^{IG}	Mean CPU time
			Mean	Max	Mean	Max	Mean	Max		
3	3	−0.51	0	0	0.06	0.08	0	0	1.68	1.83
		−0.32	0	0	0.07	0.11	0	0	1.57	1.89
		−0.15	0	0	0.04	0.06	0	0	1.77	2.10
		0	0	0	0.05	0.06	0	0	3.44	1.91
	4	−0.51	0	0	0.07	0.09	0	0	2.97	3.22
		−0.32	0	0	0.11	0.15	0	0	2.94	3.23
		−0.15	0	0	0.47	0.77	0	0	0.96	3.10
		0	0	0	0.16	0.20	0	0	2.11	3.01
	5	−0.51	0	0	0.53	0.97	0	0	2.40	4.99
		−0.32	0	0	0.36	0.66	0	0	5.14	4.97
		−0.15	0	0	0.75	1.26	0	0	2.29	4.97
		0	0	0	0.60	1.39	0	0	1.21	4.70
	6	−0.51	0	0	1.05	1.26	0	0	3.37	6.82
		−0.32	0	0	0.57	0.80	0	0	4.85	6.76
		−0.15	0	0	1.63	3.46	0	0	1.97	6.81
		0	0	0	3.15	7.85	0	0	3.50	6.51
4	3	−0.51	0	0	0.95	1.13	0	0	0.92	3.25
		−0.32	0	0	0.99	1.56	0.03	0.10	3.59	3.27
		−0.15	0	0	0.96	1.12	0	0	2.46	3.33
		0	0	0	0.60	0.70	0	0	3.10	3.03
	4	−0.51	0	0	3.64	8.47	0	0	2.23	5.46
		−0.32	0	0	8.42	21.15	0	0	2.94	5.55
		−0.15	0	0	5.81	13.90	0	0	2.50	5.50
		0	0	0	2.22	3.65	0	0	1.53	5.12
	5	−0.51	0	0	1100.71	3152.34	0	0	4.44	8.09
		−0.32	0	0.01	242.43	714.16	0	0	2.12	8.07
		−0.15	0	0	1160.62	2002.75	0	0	3.60	7.99
		0	0	0	33.51	67.57	0	0	1.62	7.60
	6	−0.51	5.67	17.00	2119.47	3600	0.02	0.07	3.23	10.93
		−0.32	11.45	20.79	2448.62	3600	0	0	1.08	10.87
		−0.15	8.53	25.58	2662.81	3600	0	0	2.28	10.88
		0	21.32	26.47	3600	3600	0	0	0.96	10.30
5	3	−0.51	0	0.01	25.99	62.12	0	0	3.15	4.72
		−0.32	0	0.01	30.14	47.32	0	0	1.80	4.71
		−0.15	0	0	80.35	170.27	0.05	0.14	0.98	4.74
		0	0	0	3.79	7.09	0	0	2.56	4.62
	4	−0.51	4.10	12.31	1883.68	3600	0.05	0.15	2.52	8.11
		−0.32	10.27	16.51	2446.64	3600	0	0	2.01	8.04
		−0.15	8.21	15.14	2582.24	3600	0	0	1.35	7.97
		0	4.75	14.24	1269.11	3600	0	0	5.23	7.69
	5	−0.51	19.35	21.05	3600	3600	−0.06	0	2.01	11.82
		−0.32	18.87	30.28	3600	3600	−0.05	0	2.02	11.73
		−0.15	28.00	36.53	3600	3600	0.10	0.29	0.72	11.72
		0	23.59	37.16	3600	3600	−0.05	0	1.85	11.05
	6	−0.51	33.70	42.78	3600	3600	0.01	0.02	1.67	15.00
		−0.32	40.27	43.96	3600	3600	−0.12	0	1.07	15.00
		−0.15	27.22	33.49	3600	3600	−0.14	0	2.41	15.00
		0	29.09	40.11	3600	3600	−0.12	0	2.59	14.77

TABLE 5. Impact of u, d and a to IG.

		M	6			8			10		
		X	8	9	10	8	9	10	8	9	10
(a)	u	$U[5,10]$	13.71	16.93	16.40	14.83	14.89	16.01	13.02	17.02	12.71
		$U[15,20]$	8.98	10.58	13.28	7.43	3.94	6.98	11.63	10.32	10.89
		$U[25,30]$	6.12	5.63	7.59	8.43	7.05	9.02	8.38	4.30	6.79
		$U[35,40]$	5.94	5.31	4.54	3.39	5.55	4.44	4.65	2.65	4.95
		$U[45,50]$	4.49	3.41	7.22	3.84	3.80	3.64	5.47	1.95	4.48
(b)	d	6	13.71	16.93	16.40	14.83	14.89	16.01	13.02	17.02	12.71
		12	13.37	14.98	13.27	13.74	7.83	10.71	9.32	11.02	12.03
		18	9.82	12.45	14.20	13.66	12.65	6.38	8.17	11.97	9.07
		24	11.59	9.00	10.07	9.43	11.39	7.81	7.81	10.45	8.83
		30	8.81	12.80	8.64	9.32	10.14	7.99	10.92	7.96	5.56
		36	10.86	9.38	8.55	10.23	7.46	7.69	8.29	7.83	8.42
(c)	a	-0.51	15.53	18.58	16.63	15.45	18.95	16.63	17.63	20.66	18.87
		-0.32	13.20	17.13	14.88	15.44	17.08	15.68	16.15	18.87	18.48
		-0.15	13.71	16.93	16.40	14.83	14.89	16.01	13.02	17.02	12.71
		0	12.58	9.14	10.38	12.41	16.95	10.02	10.20	14.03	12.32

system parameter settings on IG, more numerical experiments are conducted with different values of unit picking time and distance between adjacent workstations, including three workstation numbers $M = 6, 8$ and 10 and three X, 8, 9 and 10. Unit picking times are generated by uniform distributions $U[5,10]$, $U[15,20]$, $U[25,30]$, $U[35,40]$, and $U[45,50]$, and the distances d between adjacent workstations are $6, 12, 18, 24, 30$ and 36. Thus, we have $3 \times 3 \times 4 \times 5 \times 6 = 1080$ experimental scenarios. We generate 20 random instances for each scenario and compute the mean of RPD_s^{IG} for each experimental scenario.

Table 5(a) shows the RPD_s^{IG} values for different unit picking times. A long unit item picking time means that it takes more time to pick when the picker comes to a workstation each time. From Table 5(a), IG increases picking efficiency by 15.06% on average when unit picking time follows the distribution $U[5,10]$, and by 9.33%, 7.03%, 4.60% and 4.26% when following $U[15,20]$, $U[25,30]$, $U[35,40]$, and $U[45,50]$ respectively, which implies that IG performs better when the picking time is shorter. This is also in line with our experience that if the picker spends a long time picking at one workstation, the adjacent workstation will have enough outbound time and the probability that the adjacent workstation will allow picking increases when the picker finishes the current workstation picking task. Therefore, the simulation can also yield good results by picking at the nearest workstation. On the contrary, if the unit item picking time is short and the picker finishes the picking task at one workstation, the totes of other workstations may not be finished out of storage yet. So, the factors to be integrated when choosing the next workstation are more complex, and IG can play a greater role. In addition, some e-commerce companies sell a variety of small-sized items with short unit picking times, so it is more necessary to use IG to schedule picking.

We can see the effect of the distance between two adjacent workstations on our problem in Table 5(b), which indicates whether the picker takes a long time to reach the other workstation. Common sense suggests that if a workstation is far from the workstation where the picker is currently operating, it is undesirable to choose it as the picker's next workstation because of the large amount of walking time. The data in Table 5(b) confirm this experience, as RPD_s^{IG} is lower when d is larger, for example, IG reduces the average picking time by 15.06% when the distance between adjacent workstations is 6, compared to 11.81%, 10.93%, 9.60%, 9.13% and 8.75 when the distance is 12, 18, 24, 30 and 36. This is because when other workstations take too long to reach, the result is also satisfactory if the picker greedily chooses a nearby workstation for picking. Therefore, in this picking system, IG works better if the distance between two workstations is shorter.

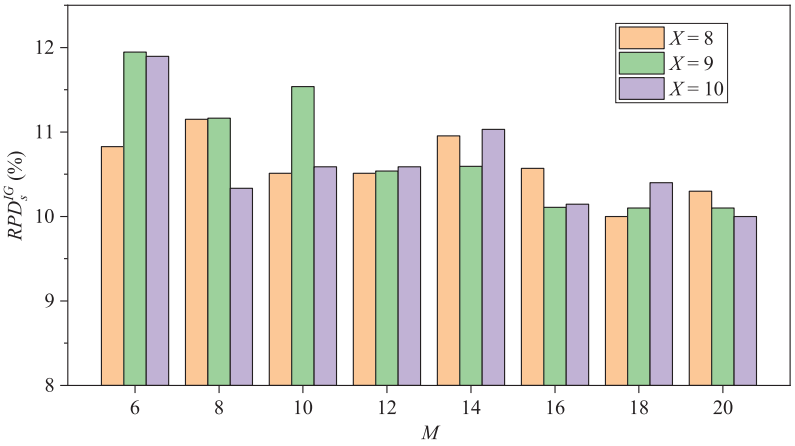

FIGURE 2. Impact of problem size to IG.

Table 5(c) shows the effect of the learning effect on our problem. We can find that when the learning effect is strong, RPD_s^{IG} is large. When $a = -0.51$, IG increases the picking efficiency by 17.66% on average, compared to 12.00% when no learning effect is considered. In addition, by comparing the average picking time with and without learning effect for each case, it is found that a large error occurs when learning effect exists for the real picking operation but are not taken into account in the calculation. This error rate is as high as about 41% when $a = -0.51$, $M = 8$ and $X = 10$. Therefore, it is necessary to consider the learning effect accurately. Moreover, when the learning effect is strong, this picking system needs IG to do picking scheduling more than depending on experience.

In order to study the trend of IG with the increase in problem size, a larger-sized experiment is performed. In the experiment, the number of workstations M is increased by 12, 14, 16, 18 and 20, and X still takes the values of 8, 9 and 10, while we set $a = -0.15$, $d = 6$. Unit picking time and outbound time of every workstation are generated from a uniform distribution $U[5, 10]$ and $U[60, 300]$, respectively. We generate 20 random instances for each scenario and compute the mean of RPD_s^{IG} for each experimental scenario. The experimental results are put in Figure 2. It can be found that the optimization effect of IG with the increase of problem size has declined, but in the picking size of 200 groups can still keep the RPD_s^{IG} greater than 10%, that is, 10% more than the simulation.

5.5. Performance of the proposed IINEH

In this part of experiments, the proposed IINEH is tested. The experiments yield the same configurations as Section 5.4 experiments, so we have made Table 6 with the same format as Table 5. The value in Table 6 is $\text{RPD}_s^{\text{IINEH}}$, where IINEH denotes our proposed heuristic, IINEH. Comparing Tables 5 and 6, we can find that the mean $\text{RPD}_s^{\text{IINEH}}$ is slightly smaller than RPD_s^{IG} in all scenarios. The average $\text{RPD}_s^{\text{IINEH}}$ of all scenarios is 9.72% while the average RPD_s^{IG} is 11.11%. However, the difference between the two methods is relatively small, and even in some scenarios, the optimization effect of IINEH is the same as that of IG, such as the scenario where u follows a uniform distribution $U[5, 10]$, $M = 10$ and $X = 8$. In addition, we record the mean CPU time of IINEH and IG for different u, d, a and problem sizes in Table 7. Table 7 shows that the CPU time of both IINEH and IG increases as the problem size increases, and they increase at a similar rate. Different u, d and a have little effect on the CPU time of IINEH and IG. But the average CPU time of IINEH is much smaller than IG, and IINEH can solve a problem in less than 1/20 of the CPU time of IG. Therefore, IINEH is an algorithm that combines the two features of optimization effectiveness and running speed to obtain a satisfactory solution quickly. It is suitable for application scenarios that require a quick solution.

TABLE 6. Impact of u, d and a to IINEH.

		M	6			8			10		
		X	8	9	10	8	9	10	8	9	10
(a)	u	$U[5,10]$	12.23	14.94	15.64	13.40	13.99	14.84	10.88	16.52	11.27
		$U[15,20]$	8.40	8.52	12.41	7.32	3.38	6.78	11.28	9.56	10.74
		$U[25,30]$	4.78	4.98	7.57	7.99	6.48	7.97	8.38	4.01	6.76
		$U[35,40]$	5.75	5.13	4.25	3.14	5.18	4.34	4.61	2.50	4.79
		$U[45,50]$	4.13	3.32	7.22	3.50	3.73	3.64	5.47	1.88	4.26
(b)	d	6	10.74	12.12	16.40	13.74	9.39	13.42	7.12	10.33	12.71
		12	12.26	14.40	12.22	12.25	6.59	9.90	7.48	9.67	11.13
		18	8.35	10.57	13.47	11.92	10.90	5.40	7.09	10.57	7.11
		24	9.57	7.44	8.85	8.21	10.58	7.05	6.36	9.41	8.34
		30	7.89	11.82	7.81	8.40	9.04	6.73	9.60	6.83	3.98
		36	10.24	8.24	7.43	8.85	5.61	6.65	7.23	7.29	7.56
(c)	a	-0.51	11.88	9.38	10.22	12.86	17.01	15.40	15.89	19.40	17.99
		-0.32	11.93	10.41	10.48	12.68	17.08	14.03	15.12	18.48	18.48
		-0.15	12.23	8.77	11.33	11.91	14.89	11.86	12.68	15.48	12.71
		0	11.02	7.93	9.57	11.38	15.91	9.38	9.37	12.36	12.32

TABLE 7. The mean CPU time of IINEH and IG (s).

M	X	IINEH			IG		
		u	d	a	u	d	a
6	8	0.39	0.38	0.37	8.74	8.68	8.74
	9	0.46	0.43	0.44	10.50	9.78	10.50
	10	0.53	0.52	0.51	10.12	10.87	10.12
8	8	0.60	0.59	0.58	14.98	15.45	14.98
	9	0.78	0.76	0.75	18.28	17.38	18.28
	10	0.95	0.95	0.94	19.87	19.34	19.87
10	8	0.96	0.96	0.93	23.91	24.11	23.91
	9	1.27	1.27	1.27	26.16	27.16	26.16
	10	1.63	1.66	1.65	29.92	30.22	29.92

5.6. Comparison IINEH and IG with the traditional algorithms

In order to compare the performance of our proposed IINEH and IG with the traditional algorithms, we designed this part of the experiment. The traditional algorithms used for comparison include the classical algorithms for solving the flowshop scheduling problem, NEH, genetic algorithm (GA), simulated annealing algorithm (SA), particle swarm optimization algorithm (PSO) and IG_NEH. Since the problem in this paper is new and there is no research on the algorithms for this problem in previous articles, the most classical form of these algorithms is used in this paper. In particular, IG_NEH is the traditional IG, which uses the classical NEH as the initial order. We refer to the literature by Ruiz *et al.* [39], Branda *et al.* [11], Liao *et al.* [25], Amirteimoori *et al.* [2], Low [27], Lin *et al.* [26], Pan and Ruiz [33] and Li *et al.* [24] for the values of the parameters of each of these algorithms. In the experiment, the number of workstations $M = 6$, 8, and 10, and X still takes the values of 8, 9 and 10, while we set $a = -0.15$, $d = 6$. For all the algorithms we have calculated its results in comparison with simulation, *i.e.*, $\mathrm{RPD}_s^{\mathrm{NEH}}$, $\mathrm{RPD}_s^{\mathrm{GA}}$, $\mathrm{RPD}_s^{\mathrm{PSO}}$, $\mathrm{RPD}_s^{\mathrm{SA}}$, $\mathrm{RPD}_s^{\mathrm{IG\_NEH}}$, the results can be seen in Table 8. We can find that the average $\mathrm{RPD}_s^{\mathrm{NEH}}$ is 6.28%, which is 3.44% smaller than that of IINEH, which indicates that our improvement plays a certain role. None of the traditional heuristic algorithms are optimized

TABLE 8. Comparison IINEH and IG with the traditional algorithms.

M	6			8			10		
X	8	9	10	8	9	10	8	9	10
RPD_s^{IINEH}	9.43	9.20	10.32	9.84	9.98	9.16	9.24	10.29	10.01
RPD_s^{NEH}	5.22	5.80	6.06	7.50	6.85	4.77	6.81	6.40	7.10
RPD_s^{IG}	10.83	11.95	11.90	11.15	11.16	10.33	10.51	11.54	10.59
RPD_s^{GA}	5.51	5.64	5.86	7.70	6.94	5.03	6.94	6.25	6.96
RPD_s^{SA}	9.53	8.93	10.27	10.69	10.09	9.51	10.10	10.33	9.87
RPD_s^{PSO}	5.19	6.11	5.84	7.66	7.15	4.92	7.09	6.64	7.35
$RPD_s^{IG\_NEH}$	9.41	9.62	10.59	10.25	9.98	9.55	9.23	10.53	10.39

as well as our proposed IG, the GA and PSO are about the same as NEH, with the average RPD of 6.32% and 6.44%, and the reason for the poor results may be that the limited running time has a greater impact on the group intelligence algorithms. SA runs better, with an average RPD_s^{SA} of 9.93%, which is more than that of IINEH, but also less than that of our proposed IG. By comparing the average RPD_s^{IG} and $RPD_s^{IG\_NEH}$, we can also find that IINEH indeed improves the optimization. Through this part of the experiment, we find that our proposed IINEH and IG can better solve the picking scheduling problem.

6. CONCLUSION

We consider a practical situation in the part-to-picker order fulfilment system where one picker is responsible for multiple workstations. We address the picking scheduling problem of this situation to minimize the maximum picking time, where learning effect and travel time between workstations are taken into account to improve scheduling accuracy. We propose two MIPs and evaluate their performance in solving this problem by small-scale numerical experiments. The rank-based model can only solve this problem for up to 9 groups, while the disjunctive model can solve up to 20 groups, so the disjunctive model performs better. For some scenarios with a large number of groups, we propose IINEH and IG, both of which are proven to be effective by numerical experiments. IINEH is well-suited for applications that prioritize speed, whereas IG is better suited for applications that prioritize accuracy. So, decision makers can choose the appropriate algorithms according to their actual needs.

In addition, the part-to-picker order fulfilment system is already computerized, and adding this scheduling function for picking is easy to implement. Therefore, in real life, it is very feasible for us to insert the IINEH or IG algorithm into the picking signal system to guide the picker to the intended workstation for picking.

However, some assumptions are made in our model. For example, the items of various types of SKUs are very different, so we suppose that they are independent of each other and there is no learning effect between them, which is not particularly realistic. This is because picking two different types of SKUs would also increase the proficiency of the picker. Therefore, it is necessary to change some assumptions in the future to further develop this study. The main contribution of this paper is not for the optimization of the algorithm, but for the scheduling problem of a novel picking scenario. Therefore, this paper does not improve the iterative greedy algorithm much, instead, more space is devoted to exploring whether the ideas of several classical algorithms are effective for solving the problem of this paper and the impact of various configurations of the system on the algorithm. So, the next research can further improve the iterative greedy algorithm to better solve the problem of this paper.

Acknowledgements. This research is supported by the Youth Foundation of Shandong Natural Science Foundation (No. ZR2022QF109).

References

[1] A.R. Ahmadi Keshavarz, D. Jaafari, M. Khalaj and P. Dokouhaki, A survey of the literature on order-picking systems by combining planning problems. *Appl. Sci.* **11** (2021) 10641.

[2] A. Amirteimoori, I. Mahdavi, M. Solimanpur, S.S. Ali and E.B. Tirkolaee, A parallel hybrid pso-ga algorithm for the flexible flow-shop scheduling with transportation. *Comput. Ind. Eng.* **173** (2022) 108672.

[3] D. Anghinolfi and M. Paolucci, A new discrete particle swarm optimization approach for the single-machine total weighted tardiness scheduling problem with sequence-dependent setup times. *Eur. J. Oper. Res.* **193** (2009) 73–85.

[4] K. Azadeh, R. De Koster and D. Roy, Robotized and automated warehouse systems: review and recent developments. *Transp. Sci.* **53** (2019) 917–945.

[5] F. Baki and R. Vickson, One-operator, two-machine open shop and flow shop scheduling with setup times for machines and maximum lateness objective. *INFOR: Inf. Syst. Oper. Res.* **41** 301–319.

[6] M.F. Baki and R.G. Vickson, One-operator, two-machine open shop and flow shop problems with setup times for machines and weighted number of tardy jobs objective. *Optim. Methods Softw.* **19** (2004) 165–178.

[7] D. Biskup, Single-machine scheduling with learning considerations. *Eur. J. Oper. Res.* **115** (1999) 173–178.

[8] D. Biskup, A state-of-the-art review on scheduling with learning effects. *Eur. J. Oper. Res.* **188** (2008) 315–329.

[9] C. Blum and A. Roli, Metaheuristics in combinatorial optimization: overview and conceptual comparison. *ACM Comput. Surv. (CSUR)* **35** (2003) 268–308.

[10] N. Boysen, R. De Koster and F. Weidinger, Warehousing in the e-commerce era: a survey. *Eur. J. Oper. Res.* **277** (2019) 396–411.

[11] A. Branda, D. Castellano, G. Guizzi and V. Popolo, Metaheuristics for the flow shop scheduling problem with maintenance activities integrated. *Comput. Ind. Eng.* **151** (2021) 106989.

[12] Ç. Cergibozan and A.S. Tasan, Order batching operations: an overview of classification, solution techniques, and future research. *J. Intell. Manuf.* **30** (2019) 335–349.

[13] T.E. Cheng, G. Wang and C. Sriskandarajah, One-operator–two-machine flowshop scheduling with setup and dismounting times. *Comput. Oper. Res.* **26** (1999) 715–730.

[14] X. Dong, H. Huang and P. Chen, An improved neh-based heuristic for the permutation flowshop problem. *Comput. Oper. Res.* **35** (2008) 3962–3968.

[15] V. Fernandez-Viagas, R. Ruiz and J.M. Framinan, A new vision of approximate methods for the permutation flowshop to minimise makespan: state-of-the-art and computational evaluation. *Eur. J. Oper. Res.* **257** (2017) 707–721.

[16] T. Franzke, E.H. Grosse, C.H. Glock and R. Elbert, An investigation of the effects of storage assignment and picker routing on the occurrence of picker blocking in manual picker-to-parts warehouses. *Int. J. Logistics Manage.* **28** (2017) 841–863.

[17] Y. Jaghbeer, R. Hanson and M.I. Johansson, Automated order picking systems and the links between design and performance: a systematic literature review. *Int. J. Prod. Res.* **58** (2020) 4489–4505.

[18] M. Ji, D. Yao, Q. Yang and T. Cheng, Machine scheduling with DeJong's learning effect. *Comput. Ind. Eng.* **80** (2015) 195–200.

[19] M. Ji, X. Tang, X. Zhang and T.E. Cheng, Machine scheduling with deteriorating jobs and DeJong's learning effect. *Comput. Ind. Eng.* **91** (2016) 42–47.

[20] P.J. Kalczynski and J. Kamburowski, On the NEH heuristic for minimizing the makespan in permutation flow shops. *Omega* **35** (2007) 53–60.

[21] C. Koulamas and G.J. Kyparisis, Single-machine scheduling problems with past-sequence-dependent setup times. *Eur. J. Oper. Res.* **187** (2008) 1045–1049.

[22] W.-Y. Ku and J.C. Beck, Mixed integer programming models for job shop scheduling: a computational analysis. *Comput. Oper. Res.* **73** (2016) 165–173.

[23] W.-H. Kuo, Single-machine group scheduling with time-dependent learning effect and position-based setup time learning effect. *Ann. Oper. Res.* **196** (2012) 349–359.

[24] X. Li, Z. Yang, R. Ruiz, T. Chen and S. Sui, An iterated greedy heuristic for no-wait flow shops with sequence dependent setup times, learning and forgetting effects. *Inf. Sci.* **453** (2018) 408–425.

[25] C.-J. Liao, C.-T. Tseng and P. Luarn, A discrete version of particle swarm optimization for flowshop scheduling problems. *Comput. Oper. Res.* **34** (2007) 3099–3111.

[26] S.-W. Lin, C.-Y. Cheng, P. Pourhejazy and K.-C. Ying, Multi-temperature simulated annealing for optimizing mixed-blocking permutation flowshop scheduling problems. *Expert Syst. App.* **165** (2021) 113837.

[27] C. Low, Simulated annealing heuristic for flow shop scheduling problems with unrelated parallel machines. *Comput. Oper. Res.* **32** (2005) 2013–2025.

[28] A.S. Manne, On the job-shop scheduling problem. *Oper. Res.* **8** (1960) 219–223.

[29] M. Masae, C.H. Glock and E.H. Grosse, Order picker routing in warehouses: a systematic literature review. *Int. J. Prod. Econ.* **224** (2020) 107564.

[30] G. Mosheiov and J.B. Sidney, Scheduling with general job-dependent learning curves. *Eur. J. Oper. Res.* **147** (2003) 665–670.

[31] M. Nawaz, E.E. Enscore Jr. and I. Ham, A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega* **11** (1983) 91–95.
[32] D. Okołowski and S. Gawiejnowicz, Exact and heuristic algorithms for parallel-machine scheduling with DeJong's learning effect. *Comput. Ind. Eng.* **59** (2010) 272–279.
[33] Q.-K. Pan and R. Ruiz, A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime. *Comput. Oper. Res.* **40** (2013) 117–128.
[34] J. Pei, X. Liu, P.M. Pardalos, K. Li, W. Fan and A. Migdalas, Single-machine serial-batching scheduling with a machine availability constraint, position-dependent processing time, and time-dependent set-up time. *Optim. Lett.* **11** (2017) 1257–1271.
[35] J. Pei, X. Liu, P.M. Pardalos, A. Migdalas and S. Yang, Serial-batching scheduling with time-dependent setup time and effects of deterioration and learning on a single-machine. *J. Global Optim.* **67** (2017) 251–262.
[36] J. Pei, B. Cheng, X. Liu, P.M. Pardalos and M. Kong, Single-machine and parallel-machine serial-batching scheduling problems with position-based learning effect and linear setup time. *Ann. Oper. Res.* **272** (2019) 217–241.
[37] I. Ribas, R. Companys and X. Tort-Martorell, Comparing three-step heuristics for the permutation flow shop problem. *Comput. Oper. Res.* **37** (2010) 2062–2070.
[38] R. Ruiz and T. Stützle, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *Eur. J. Oper. Res.* **177** (2007) 2033–2049.
[39] R. Ruiz, C. Maroto and J. Alcaraz, Two new robust genetic algorithms for the flowshop scheduling problem. *Omega* **34** (2006) 461–476.
[40] E. Vallada and R. Ruiz, A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *Eur. J. Oper. Res.* **211** (2011) 612–622.
[41] T. Van Gils, K. Ramaekers, A. Caris and R.B. de Koster, Designing efficient order picking systems by combining planning problems: state-of-the-art classification and review. *Eur. J. Oper. Res.* **267** (2018) 1–15.
[42] H.M. Wagner, An integer linear-programming model for machine scheduling. *Naval Res. Logistics Q.* **6** (1959) 131–140.
[43] J.-B.Wang, Single-machine scheduling with past-sequence-dependent setup times and time-dependent learning effect. *Comput. Ind. Eng.* **55** (2008) 584–591.
[44] T.P. Wright, Factors affecting the cost of airplanes. *J. Aeronautical Sci.* **3** (1936) 122–128.
[45] C.-C. Wu, Y. Yin, W.-H. Wu, H.-M. Chen and S.-R. Cheng, Using a branch-and-bound and a genetic algorithm for a single-machine total late work scheduling problem. *Soft Comput.* **20** (2016) 1329–1339.
[46] D. Yang, Research on Intelligent Logistics Warehousing System Design and Operation Strategy Optimizaion. Economy & Management Publishing House (2023).
[47] Y. Yin, C.-C. Wu, W.-H. Wu and S.-R. Cheng, The single-machine total weighted tardiness scheduling problem with position-based learning effects. *Comput. Oper. Res.* **39** (2012) 1109–1116.
[48] Y. Yin, W.-H. Wu, W.-H. Wu and C.-C. Wu, A branch-and-bound algorithm for a single machine sequencing to minimize the total tardiness with arbitrary release dates and position-dependent learning effects. *Inf. Sci.* **256** (2014) 91–108.

Please help to maintain this journal in open access!

This journal is currently published in open access under the Subscribe to Open model (S2O). We are thankful to our subscribers and supporters for making it possible to publish this journal in open access in the current year, free of charge for authors and readers.

Check with your library that it subscribes to the journal, or consider making a personal donation to the S2O programme by contacting subscribers@edpsciences.org.

More information, including a list of supporters and financial transparency reports, is available at https://edpsciences.org/en/subscribe-to-open-s2o.