Real-time order picking of a robotic put wall: a simulation-based metaheuristic optimisation

Jianbin Xin

School of Electrical and Information Engineering,
Zhengzhou University,
No. 100 Science Road, Zhengzhou 450001, China
and
State Key Laboratory of Intelligent Agricultural Power Equipment,
No. 39, Xiyuan Road,
Luoyang 471039, China
Email: j.xin@zzu.edu.cn

Ziyuan Kang

School of Electrical and Information Engineering,
Zhengzhou University,
No. 100 Science Road, Zhengzhou 450001, China
Email: 704058543@qq.com

Andrea D'Ariano

Department of Civil, Computer Science and
Aeronautical Technologies Engineering,
Roma Tre University,
00146 Roma, Italy
Email: andrea.dariano@uniroma3.it

Lina Yao*

School of Electrical and Inforamtion Engineering,
Zhengzhou University,
No. 100 Science Road, Zhengzhou 450001, China
Email: yaoln@zzu.edu.cn
*Corresponding author

Abstract: A robotic put wall has the potential to significantly enhance picking productivity in the logistics industry. This paper introduces a new computational method for scheduling a robotic put wall system that processes randomly arriving items. The method comprises a simulation-based model and a customised metaheuristic that optimises performance at regular intervals. The simulation model is developed using advanced discrete-event software that can include operational details of the picking process. The genetic algorithm with a new encoding scheme is tailored to solve the combinatorial optimisation problem of determining the appropriate destinations. To evaluate the proposed method, case studies based on real-world applications in a put wall manufacturing company were used. The method outperforms three rule-based real-time scheduling methods, as demonstrated by the results. Moreover, the integrated approach can determine the minimum number of vehicles required.

Keywords: order picking system; robotic put wall; real-time scheduling; simulation-based optimisation.

Reference to this paper should be made as follows: Xin, J., Kang, Z., D'Ariano, A. and Yao, L. (2024) 'Real-time order picking of a robotic put wall: a simulation-based metaheuristic optimisation', *Int. J. Automation and Control*, Vol. 18, No. 5, pp.516–536.

Biographical notes: Jianbin Xin received his BSc in Electrical Engineering from the Xidian University, China in 2007, MSc in Control Science and Engineering from the Xi'an Jiaotong University, China in 2010, and PhD from the Department of Maritime and Transport Technology, Delft University of Technology, The Netherlands in 2015. Currently, he is an Associate Professor at the Department of Automation, Zhengzhou University, China. His research interests include planning and control of smart logistics systems and cooperative robots.

Ziyuan Kang received his Bachelor's degree and Master of Engineering from the School of Electrical Engineering at Zhengzhou University in 2019 and 2023, respectively. His research interests include simulation and optimisation for smart logistics.

Andrea D'Ariano received his BS and MS in Computer Science, Automation, and Management Engineering from the Roma Tre University and PhD from the Department of Transport and Planning, Delft University of Technology, in April 2008, under the supervision of Professor I.A. Hansen. Currently, he is a Professor at the Department of Engineering, Roma Tre University. His research interests include the study of scheduling problems with application to public transportation and logistics. He is an associate editor of well-known international journals, such as *Transportation Research – B: Methodological*, *Transportation Research – C: Emerging Technologies*, and *Transportation Research – E: Logistics and Transportation Review.*

Lina Yao received his PhD in Control Theory and Control Engineering from the Institute of Automation, Chinese Academy of Sciences, Beijing, China in 2006. Then, she joined Zhengzhou University, Zhengzhou, China, where she is currently a Professor. From September 2007 to March 2008, she was a research fellow at the University of Science and Technology of Lille in France. Her research interests are in fault diagnosis and fault-tolerant control for dynamic systems, stochastic distribution control, and flight control.

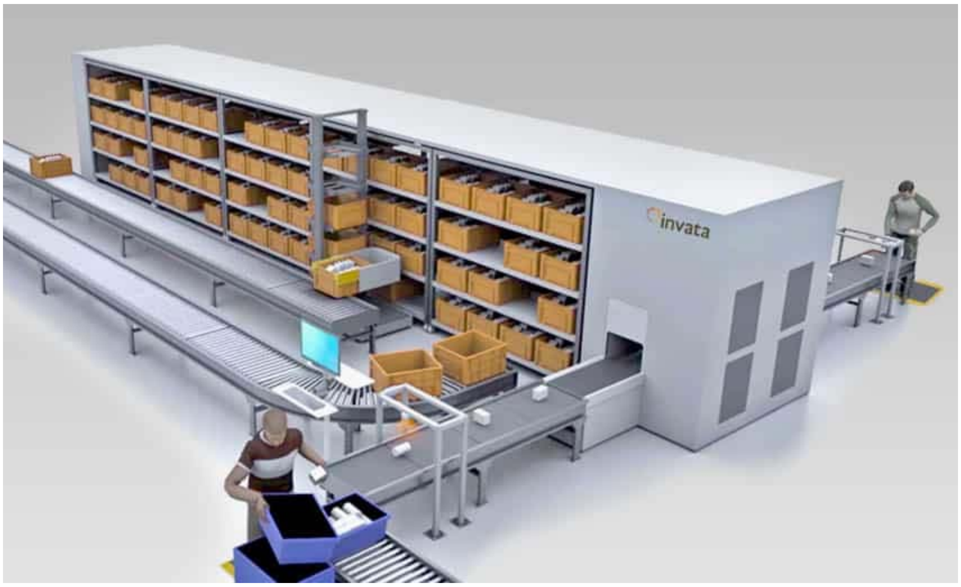
1 Introduction

With the rapid growth of e-commerce, there has been a significant increase in the number of parcels being delivered from warehouses to customers. As a result, the order picking system (OPS) has become an essential component of the logistics process. Order picking involves selecting products from the warehouse and packing them into parcels based on customer orders (Boysen et al., 2021; Tan et al., 2021). The parcels are then sorted by destination and prepared for shipment. Order picking is widely regarded as the most labour-intensive operation in the warehouse, and improving productivity in this area is a top priority for warehousing professionals (de Koster et al., 2007).

Modern order picking systems are faced with the challenge of handling a high volume of small orders with tight delivery deadlines (Boysen et al., 2019b). Meeting these demands while maintaining high-performance levels can be difficult for OPSs (Azadeh et al., 2019). One solution for increasing productivity and reducing labour costs is to automate the order picking process. Various automated order picking systems, such as automated guided vehicles (AGVs), rail guided vehicles (RGV), and other types of robots, have been investigated (Sun et al., 2021; Luo et al., 2023; Lamballais et al., 2017; Xin et al., 2014). In this paper, we focus on improving the performance of a robotic put wall, an automated OPS that processes a high volume of orders in a limited space using a fleet of RGVs (see Figure 1). The robotic put wall enables fast, automated pick-up and sortation of discrete order consolidation items.

The operations of a robotic put wall are complex and challenging. Processing a large number of orders quickly is a primary requirement. When multiple goods and orders are provided, different goods must be packed into the same container for each order, and the destination sequence must be optimised. Additionally, the RGV must operate in a collision-free environment. Furthermore, goods arrive randomly at the robotic put wall, and detailed information is not available until they are scanned. Given these complexities, we propose a novel integrated computational method to increase the productivity of the robotic put wall.

Figure 1 Schematic layout of a put wall (see online version for colours)



Source: Courtesy of Invata (Invata Automated Robotic Put Wall Solutions, 2023)

1.1 Related work

This section reviews the literature on the planning problem for the robotic put wall. OPS planning can be categorised into three areas: strategic, tactical, and operational decision problems (van Gils et al., 2018). Strategic problems focus on long-term competitive strategies, such as layout design and material handling equipment selection (van Gils et al., 2018). Tactical problems address medium-term decisions, such as resource dimension, zoning, and storage assignment (Yu and De Koster, 2009; Lee et al., 2020). Operational decision problems focus on daily operations, such as batching, routing, and job assignment, to maximise productivity (Wagner and Mönch, 2022; Bódis and Botzheim, 2018).

Given our research focus on real-time scheduling of the robotic put wall, we will review the literature on operational decision problems for OPS and the put wall.

1.1.1 OPS's operational decisions

This section provides an overview of the literature on real-time scheduling for the robotic put wall. Real-time scheduling is concerned with operational decisions such as order batching and routing.

Order batching policies determine which customer orders should be combined into a single pick round (van Gils et al., 2018). The order batching problem is investigated using return, midpoint, and traversal routing strategies (Öncan, 2015). The author presents mixed integer linear programming (MILP) formulations for these strategies, along with an iterative local search algorithm. Gil-Borrás et al. (2021) focus on the online order batching processing problem with multiple pickers and propose a multi-start procedure hybridised with a variable neighbourhood descent metaheuristic to solve it. Overall, these studies contribute valuable insights into order batching policies and provide different strategies and algorithms to tackle the order batching problem.

Routing policies dictate the sequence of storage locations to be visited in each pick round to fulfil a pick list (van Gils et al., 2018). Chabot et al. (2017) investigate the order-picking routing problem with weight, fragility, and category constraints. They propose two mathematical models and employ five heuristic methods, including extensions of a classical large gap, midpoint, S-shape, and combined heuristics. Matusiak et al. (2014) tackle the order picking and picker-routing combination problem in the warehouse using the simulated annealing algorithm. They estimate cost savings by batch processing more than two customer orders to eliminate unnecessary routing. Masae et al. (2021) propose an efficient optimal order picker routing policy for the leaf warehouse using an Eulerian graph and dynamic programming procedure. They also develop four simple routing heuristics.

Order pickers are often required to retrieve orders within tight time windows. The job assignment planning problem determines the optimal order in which orders or batches of orders should be retrieved, as well as how orders should be assigned to a limited number of pickers (van Gils et al., 2018). Henn (2015) emphasises the significance of retrieving customer orders within their specified time windows and propose a local search-based method to minimise the total delay of a given set of orders. In a related study, Henn and Schmid (2013) demonstrate how metaheuristics can be utilised to minimise the total tardiness of a given set of customer orders.

Based on the available literature, it appears that researchers have primarily focused on the process of orders from storage to the picking station, rather than the operational decision during order picking. Therefore, it is worthwhile to investigate the operational decision in the picking station's working process.

1.1.2 Put wall system

A put wall is a piece of hardware made out of containers that are mostly used for order picking. Put walls can manage a high volume of orders while maintaining a modest footprint. This order-picking mechanism not only saves time on item selection but also makes the best use of the warehouse's limited area. As a result, many online retailers in the medical and fresh food industries use placed barriers to pick orders (Boysen et al., 2019a).

Put walls receive little attention in comparison to the more commonly used OPS, such as RMFS and AS/RS. Ardjmand et al. (2019) study the integrated order batching and picker routing problem, in which two genetic algorithms (GA) with customised operators and list-based simulated annealing are proposed. The manual put wall's batched order bin sequencing problem is investigated, to quickly pick orders into the put wall and avoid packers' unproductive idle time (Boysen et al., 2019b). In Ardjmand et al. (2020), a mixed integer programming model for sequential batching and routing in a put wall-based picking system is proposed. To minimise the total travel time and the maximum time, they study the order batching and picker routing problem of a put wall-based OPS and propose a coevolutionary genetic algorithm and an archived multi-objective simulated annealing.

In summary, it is observed that all the existing research focuses on the manual put wall. The research on the robotic put wall has not been investigated. Consequently, it is unclear how the robotic put wall is scheduled.

1.1.3 Contributions and structure

The main contribution of this paper is to achieve the dynamic order picking of the robotic put wall system, which has not been investigated, to the best of our knowledge. We propose a novel simulation-based method to dynamically optimise the performance of the robotic put wall system. Our proposed scheduling approach incorporates simulation and genetic algorithm, which enables the system to dynamically select the destinations of randomly arriving goods. Importantly, dynamic scheduling occurs in real time, ensuring the system is always operating at peak performance.

The rest of this paper is structured as follows: Section 2 introduces the problem statement as well as the main steps for constructing the discrete event simulation. Section 3 proposes a simulation-based dynamic scheduling method that combines the genetic algorithm and simulation. In Section 4, we conduct case study experiments to discuss its operational performance and determine the optimal number of RGVs. Section 5 summarises the paper and suggests future research directions.

2 Simulation-based modelling

This section outlines the problem statement and introduces the simulation-based method to simulate the robotic put wall system. The first part defines the research problem of the put wall and the second part provides its modelling framework using simulation.

2.1 Problem statement

Consider the robotic put wall system depicted in Figure 1, which consists of a series of dedicated shelving. Put walls can handle a high volume of orders in a small amount of space. A fleet of RGVs is used in the robotic put wall system to load a batch of goods from the conveyor and then unload them into different boxes linked to different customers for packing. Several good containers are available in front of a worker. When goods arrive at the robotic put wall, they are manually scanned and their type information is determined. Based on the order requirements, the goods are transported by the RGVs and placed in the appropriate container. When the order is completed, the container will be manually removed from the automated put wall and packed. After that, the empty container will return to the automated put wall and participate in the next round of order picking.

In our research problem, we focus on scheduling batch picking operations by the put wall system, i.e., how to schedule the RGV for transporting put goods with unknown order information and how to allocate these goods to form the required orders.

Before modelling the put wall system, important assumptions are made below.

- each RGV can only load one good at a time
- the order of the goods is unknown, and the good information is not available until the goods are manually scanned
- each RGV starts the next task after completing the current task
- all RGVs start from the bottom track
- the goods destination, i.e., the goods box, is located on the vertical track of the put wall
- each RGV stops when performing the loading and unloading operations
- the put wall system stops until all the orders are completed.

2.2 Discrete event simulation-based modelling

Discrete-event simulation (DES) is regarded as an efficient computer tool to model, optimise and analyse real-world discrete event systems, which are widely used for production scheduling, traffic management, and resource utilisation problems (van Vianen et al., 2016; Fabri et al., 2022).

In DES, the system's state changes only during a specific time period defined as an event. DES can be expressed as a set of events on the timestamp, where different events represent the system's states at different times. The simulation based on discrete events must accurately represent the system and reflect its overall operation logic. As a result, we divide the modelling process into the following steps based on discrete events:

1 Data collection: Analyse the actual system and collect data related to the model's establishment, such as the system's size and the parameters of the system's corresponding modules, such as length, height, speed, and so on.

2 Tool selection: Choose appropriate software tools to simulate the actual system, simulate and restore the system's operation logic and constraints, and realise data input and data collection functions.

3 Create the model: Before data collection and simulation optimisation, it is essential to confirm through specific data sets that the model's performance closely matches the real system. Restart at Step 2 if the model's performance significantly differs from that of the actual system. Figure 2 illustrates the logic diagram of the DES model developed in this work.

4 Simulation and results analysis: Based on the validity of the simulation results, determine whether there is room for improvement in the model and propose an improvement path.

Figure 2 Operation logic diagram of the RGV-based robotic put wall

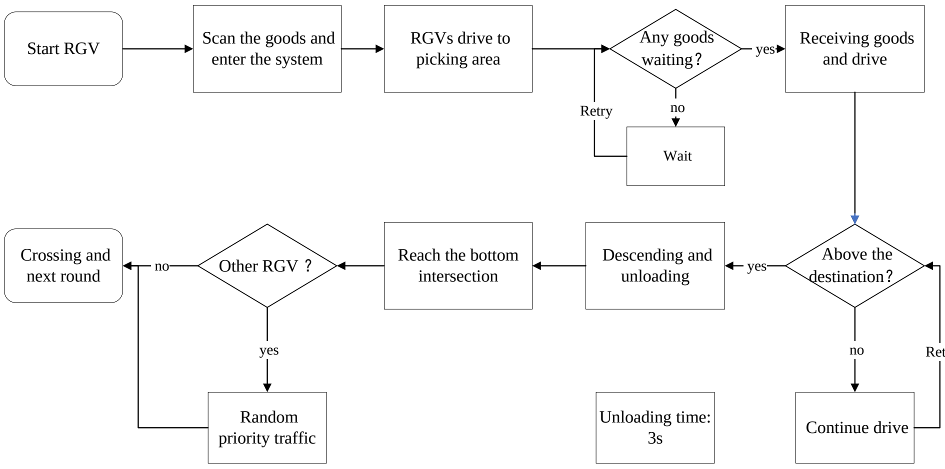
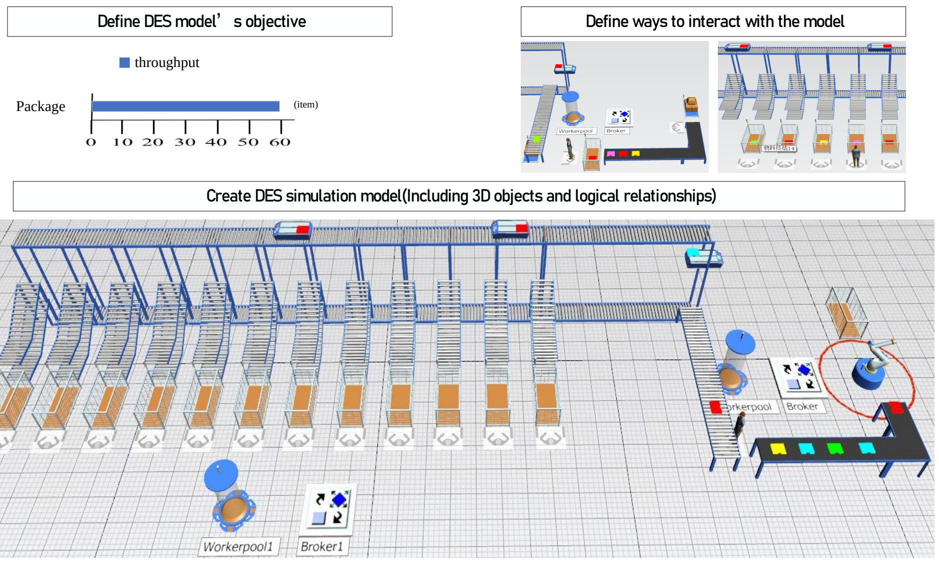


Figure 2 provides the logic diagram of the developed DES model, in which all RGVs are processed in parallel. First, all the empty RGVs are parked on the bottom track of the system before starting to load the goods. When a certain good is available, the good is manually scanned and then transported from the scanning point to the unloading point via the belt conveyor. In this circumstance, after loading the good, the RGV moves to the top horizontal track and checks the destination shelf when passing the intersection. The RGV moves downside when the destination shelf is detected. After unloading the good, the RGV reaches the intersection of the vertical track and the bottom horizontal track, and join the RGV queue for loading new goods. When two RGVs approach the same intersection in the bottom horizontal track, the conflict is resolved by a prioritised rule. After passing the intersection, the RGV arrives at the loading location again and starts the next cycle.

Table 1 Indexes and sets

Notation	*Description*
n	Total number of RGVs
k	Total number of RGVs in S_{pool}
V	Set of RGVs, $V = [1, 2, ..., V_n]$
L	Location of RGVs, $L = [L_1, L_2, ..., L_n]$
D	Module of unloading shelf, include A, B, C, ...
d	Destination module of RGV, with $d \in D$
F	Floor of unloading place
f	Destination floor of RGV, with $f \in F$
G	Set of goods categories, including a, b, c, d, and e
g	Type of goods loaded by the RGV, with $g \in G$
S_{pool}	Pool composed of RGVs participating in online scheduling

Figure 3 Implementing the robotic put wall by using Tecnomatix Plant (see online version for colours)


Define DES model' s objective
throughput
Package
(item)
0 10 20 30 40 50 60
Define ways to interact with the model
Workerpool
Broker
Create DES simulation model(Including 3D objects and logical relationships)
Workerpool1
Broker1

In this paper, we use the state-of-the-art simulation software Tecnomatix Plant Simulation (Bangsow, 2020), which enables us to simulate, analyse and schedule the order-picking process of the robotic put wall. Specifically, Plant allows us to create another virtual system potentially used for digital twins and optimise the picking process by using the customised toolbox. Table 1 defines the used defined indexes and sets. Figure 3 gives the robotic put wall implemented by Tecnomatix Plant.

3 Simulation-based dynamic scheduling

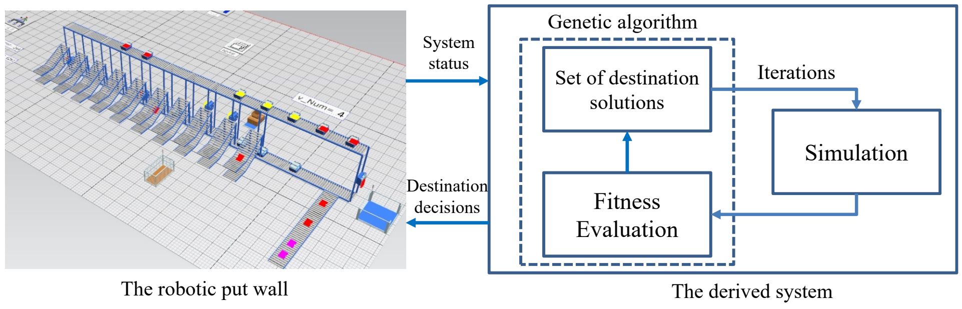
In this section, for the robotic put wall, using the advantages of fast and high accuracy in solving problems based on metaheuristic algorithms (Laha et al., 2010), we propose a dynamic scheduling method, which is based on the simulation model and a metaheuristic. The scheduling framework is introduced first, followed by a discussion of a customised scheduling algorithm.

3.1 Framework

Figure 4 gives the real-time scheduling framework for the considered put wall. In this framework, a virtual representation is created to interact with the simulated physical system, forming a derived system to optimise the order-picking process in real-time.

The physical system uses sensors to identify all RGVs in the simulation and transmits the information (RGV location and destination, RGV index, and good type) to the derived virtual system. The derived system then formulates the optimisation problem and uses the tailored GA module to discover the optimal solution. Following a return to the physical system of the best choice, the unloaded RGVs look for the corresponding unloading point as their destination. According to the most recent information, every RGV is still in operation. The processing processes of the physical system can be quickly optimised thanks to the developed derived model and the tailored GA module.

Figure 4 Framework of simulation-based dynamic scheduling (see online version for colours)



3.2 Customised scheduling

This part introduces the developed dynamic scheduling algorithm by using the created derived model and the customised GA inside the simulation software Tecnomatix Plant. On the one hand, GA has been proven to be a simple but powerful metaheuristic to solve practical combinatorial optimisation problems (Xin et al., 2022, 2023). The GA has a relatively simple algorithmic structure, but this metaheuristic has a good ability to diversify the search in the feasible region of the search space. On the other hand, Tecnomatix Plant provides an easy-to-use module to encode the feasible solutions of the formulated order-picking problem.

Genetic algorithms are random search techniques based on natural selection (Cui, 2020). GA in general uses a set of feasible solutions, which is defined as a population, and iteratively finds new high-quality solutions through the following customised operation: selection, crossover, and mutation. The selection operation is to pick up high-quality solutions from the current population, to allow convergence toward optimal solutions. The crossover and mutation operations generate new solutions to enlarge the diversity of the population to accelerate the convergence of the algorithm, i.e., improving local optimal solutions.

In the next parts, we discuss the customised GA's essential elements, including encoding, three operations, and fitness functions. We will also present the detailed scheduling algorithm procedures.

3.2.1 Encoding

In GAs, a population (a set of possible solutions) needs to be initialised and further optimised. The encoding scheme for each solution is highly relevant to the solution quality. For the considered order-picking process, we consider the permutation encoding scheme, in which each chromosome contains the destinations of the good.

Since both the type and destination of every picked good can be different, we propose a two-dimensional (2D) encoding scheme rather than the one-dimensional (1D) encoding scheme used for common scheduling problems. The good type information and the destination are included in these two dimensions, respectively. Each chromosome, which is defined as X, contains X_g and X_d in two dimensions representing the good type and the destination information.

Figure 5 Illustration of the proposed 2D encoding scheme

Dimension1	a	b	e	c	a	b	c	d
Dimension2	1	2	4	3	2	1	4	3

Then, Figure 5 illustrates the detailed encoding method based on the defined symbols earlier. In this example, eight goods are listed in order. Dimensions 1 and 2 correspond to the content of X_g (good category) and X_d (good destination). The categories of these eight goods are $\{a, b, e, c, a, b, c, d\}$, and their destinations are listed as $\{1, 2, 4, 3, 2, 1, 4, 3\}$.

In the proposed re-scheduling algorithm, the solution only changes in the destination when involving the same type of cargo. The following part discusses how these solution changes using the GA's operations to improve the solution quality.

3.2.2 Pseudocode

The pseudocode of the proposed dynamic scheduling algorithm is shown in Algorithm 1. Initially, the positions of all the RGVs are reported as inputs, and the RGVs start to load the goods waiting in the conveyor. At each instant t, for each RGV carrying a particular good, detect if the RGV is moving downward in the column. These goods that are unsatisfying this condition are included in a non-empty decision pool S_{pool} used for solving an optimisation problem using the simulation. If the decision pool S_{pool} is non-empty, a derived system is created to simulate the original put wall. In such a derived system, GA is customised to optimise the destinations of the picked goods by using selection, crossover, and mutation operations. These operations are discussed in the following parts.

Algorithm 1 Simulation-based real-time scheduling algorithm

```
Initialise the locations of all the RGVs: $L_1$, $L_2$, ..., $L_n$
for $t = 0, 1, 2, ..., t_{\max}$ do
    Detect the status of each RGV and compose the decision pool $S_{\text{pool}}$
    while $S_{\text{pool}}$ is not empty do
        Derive the system of the put wall
        Initialise $P(iter)$
        while $iter \leq iter_{\max}$ do
            for $i = 1 : N_{\text{p}}$ do
                Evaluate the fitness $F(X)$ of each solution in population $P(iter)$
            end for
            Selection over $P(iter)$ and form $P_{\text{new}}(iter)$
        end while
        for $i = 1 : N_{\text{p}}$ do
            Randomly choose two solutions from $P_{\text{new}}(iter)$ as $c_i^1(iter)$ and $c_i^2(iter)$
            Crossover over $c_i^1(iter)$ and $c_i^2(iter)$ as $c_i(iter)$ with a probability $\alpha$
            Mutation over $c_i^1(iter)$ or $c_i^2(iter)$ as $c_i(iter)$ with a probability $\beta$
            Report $c_i^1(iter)$ as $c_i(iter)$ with a probability $(1 - \alpha - \beta)$
        end for
        $P(iter + 1) = c_1(iter) \cup c_2(iter) ... \cup c_N(iter)$
    end while
    Wait for the next moment $t$
end for
```

3.2.3 GA operations

Three operations (selection, crossover, and mutation) of GA are detailed in this part. These operations are essential elements of GA to generate new and potentially better individuals during several iterations.

The selection operation is based on a roulette wheel selection strategy, as suggested by Kramer (2017). In this strategy, the fitness $f(x_i)$ of each x_i in the population P is needed. The probabilities of each individual being inherited by the next generation population (defined as p_i) and the cumulative probability of chromosome (defined as q_i), are used to select the next generation. p_i and q_i) are calculated as follows:

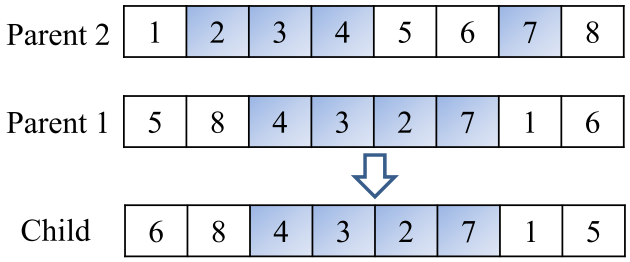
$$p_i = \frac{f(x_i)}{\sum_{j=1}^{M} f(x_j)} \tag{1}$$

$$q_i = \sum_{j=1}^{i} p(x_j) \tag{2}$$

When p_i and q_i are calculated, generate a uniformly distributed random number r ($r \in [0, 1]$) for each individual x_i one by one. If $r < q_i$, select the individual x_i, otherwise choose the individual x_k which satisfies the condition $q_{k-1} < r \leq q_k$.

For the crossover operation, the order crossover strategy is employed to generate new individuals in the next generation. This commonly-used strategy is simple and efficient for permutation-encoded chromosomes, as suggested by Kramer (2017). Under this strategy, the crossover operation is performed based on two chromosomes randomly chosen from the updated population after the above selection operations. The chosen two random chromosomes are regarded as parents 1 and 2. Two random crossover points are created, and the destination numbers between these two points are copied from parent 1 to a new chromosome in the same positions. Then, remove these copied numbers from parent 2 and insert the remaining arc numbers in parent 2 into the new chromosome in order from the second crossover point. When the second half of the new chromosome is filled, fill it from the first half. The crossover process is illustrated in Figure 6.

Figure 6 Illustrative order crossover operation (see online version for colours)



Regarding the mutation operation, the swap strategy is employed because it provides a new order but makes small changes to the existing order of the put wall. In this strategy, two positions in the chromosome are selected randomly, and their destination numbers are swapped.

3.2.4 *Fitness function*

In the offline scheduling problem, the objective is to minimise the total time for the put wall to complete picking the last good (makespan). In our online scheduling sub-problem, the objective of the local optimisation problem is transformed into the sum of the time when each RGV reaches the loading place. The reason is that if the makespan is set to be objective, there may exist a delay for new goods when being to be loaded.

In the customised GA, we let the fitness function written as follows:

$$F(X) = \sum_{i=1}^{n} T(V_i) \tag{3}$$

where $T(V_i)$ is the time (in the derived system) when RGV V_i reaches the loading place after completing the unloading task. Note that $T(V_i)$ are obtained by simulating all the transport tasks in the derived system. For a large-scale put wall, the computation burden can be large when considering a large population P for GA. In the following section, we will discuss the proper parameters of the GA applicable for real-time scheduling.

4 Experimental results

This section conducts extensive simulation experiments and discusses the results of the proposed real-time scheduling method against two commonly-used methods. We first present the setting of simulation experiments and then discuss the experimental results.

4.1 Experiment setup

This part introduces the experimental settings of the robotic put wall simulation system. Subsection 4.1.1 introduces the scenarios generation and overall setting and Subsection 4.1.2 discusses the parameter selection of the proposed scheduling algorithm suitable for the put wall.

4.1.1 Scenarios generation and overall setting

Here we compare the proposed dynamic scheduling method to three rule-based methods in practice to validate the feasibility of the methods proposed in this study.

We consider a real put wall of 12 columns, and two order boxes are located for each column. The discrete matrix between different columns is given in Table 2. We consider four scenarios with different numbers of items and arrival rates. Scenarios 1 and 2 have a total of 120 goods and 240 goods, and each item arrives every second. Scenarios 1 and 2 include a total of 120 goods and 240 goods, and each item arrives within a random time interval between one second and two seconds. The good type and destination of all the goods are randomly generated in an unknown order. These data are derived from real-world applications at a put wall manufacturer company.

Table 2 Distance matrix between different columns (unit: metre)

Column	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
1	0	2	4	6	8	10	12	14	16	18	20	22
2	2	0	2	4	6	8	10	12	14	16	18	20
3	4	2	0	2	4	6	8	10	12	14	16	18
4	6	4	2	0	2	4	6	8	10	12	14	16
5	8	6	4	2	0	2	4	6	8	10	12	14
6	10	8	6	4	2	0	2	4	6	8	10	12
7	12	10	8	6	4	2	0	2	4	6	8	10
8	14	12	10	8	6	4	2	0	2	4	6	8
9	16	14	12	10	8	6	4	2	0	2	4	6
10	18	16	14	12	10	8	6	4	2	0	2	4
11	20	18	16	14	12	10	8	6	4	2	0	2
12	22	20	18	16	14	12	10	8	6	4	2	0

The proposed method is compared with three rule-based dynamic scheduling methods, i.e., random, minimal distance (MD), and congestion-aware. These three compared method are summarised as follows:

- the random method randomly select the destinations of each good when perform the dynamic scheduling

- the MD method selects the nearest destination of the good, when multiple orders of the same type of good are received
- congestion-aware methods avoid selecting the nearest destination column, which may cause congestion when multiple RGVs are involved.

The makespan and computation time (CT) are both recorded to compare the performances. Tecnomatix Plant 16.0 is used to simulate the robotic put wall. The hardware for all these experiments is an Intel i7-9700 processor (3.0 GHz) with 8 GB of memory.

4.1.2 Algorithm parameter setting

In the designed GA, four key parameters (the population size N_p, the maximal number of iterations $iter_\text{max}$, the probabilities of crossover and mutation P_c and P_m) need to be selected. First, we conducted a group of numerical experiments via the proposed dynamic scheduling method to better set N_p and $iter_\text{max}$, under a commonly used setting ($P_\text{c} = 0.8$, $P_\text{m} = 0.1$) (Kramer, 2017). The average fitness value and computation time are presented in Table 3 (120 goods, $N_k = 5$).

Table 3 Averaged fitness value and computation time for setting the GA's parameters

N_p	$iter_{max}$	*Fitness*	*CT (unit: ms)*
50	10	124.095	69.7
50	20	119.52	72.1
50	50	119.52	301.7
50	100	119.52	757.6
20	10	127.655	52.3
20	20	121.015	72.1
20	50	119.52	201.5
20	100	119.52	395.7

Table 3 shows the average fitness and computation time for two different parameter settings. The N_p and $iter_\text{max}$ values chosen are 50 and 20, respectively. This setting is recommended because it produces the smallest fitness values in the shortest amount of time.

Then, under the setting ($N_\text{p} = 50$ and $iter_\text{max} = 20$), P_c and P_m are varied to see their influences on the fitness value over the iterations, as shown in Figure 7. The compared results in Figure 7 show that the setting ($P_\text{c} = 0.8$, $P_\text{m} = 0.1$) indeed gives the lowest fitness value with a fast convergence.

4.2 Result discussion

Tables 4–7 compare the computational performances of our proposed simulation-based scheduling method against three practical rule-based real-time scheduling methods (random, congestion-aware, and MD) for the four scenarios. The number of RGVs N_k varied for each scenario to verify the robustness of the proposed method.

Table 4 Compared computational results with respect to scenario 1

Settings	*Random*		*Congestion aware*		*MD*		*Our proposed*	
N_k	*makespan (s)*	*CT (ms)*	*makespan (s)*	*CT (ms)*	*makespan (s)*	*CT (ms)*	*makespan (s)*	*CT (ms)*
7	691.82	0.4	692.16	0.8	696.54	2.6	*668.81*	289.1
8	616.12	0.5	615.12	1.2	619.31	3.5	*594.77*	337.0
9	558.98	0.6	555.94	1.8	558.26	4.4	*537.39*	369.1
10	509.47	0.8	507.91	2.7	507.81	5.1	*490.26*	418.5
11	479.55	1.0	475.79	3.8	478.23	6.0	*458.53*	477.3
12	449.23	1.3	444.25	4.9	444.77	7.0	*431.72*	561.2
13	421.12	1.5	416.03	6.2	415.09	7.8	*404.20*	642.3
14	405.07	1.9	399.81	7.3	398.59	8.8	*387.18*	757.6
15	389.58	2.3	383.36	8.7	383.11	9.7	*373.34*	812.3
16	390.85	2.6	384.25	9.9	383.15	10.5	*377.55*	887.4
17	396.97	3.1	389.13	11.4	391.27	12.1	*383.27*	957.9
18	408.05	3.6	400.24	12.8	403.09	13.6	*396.91*	1,029.8
Average	476.40	1.6	472.00	6.0	473.10	7.6	*458.66*	628.3

Table 5 Compared computational results with respect to scenario 2

Settings	*Random*		*Congestion aware*		*MD*		*Our proposed*	
N_k	*makespan (s)*	*CT (ms)*	*makespan (s)*	*CT (ms)*	*makespan (s)*	*CT (ms)*	*makespan (s)*	*CT (ms)*
7	693.08	0.3	694.74	0.8	697.66	2.5	*667.73*	293.7
8	617.72	0.4	617.01	1.2	618.23	3.5	*595.60*	341.0
9	560.35	0.6	557.55	1.7	559.92	4.4	*538.04*	373.5
10	511.63	0.8	509.79	2.7	509.56	5.0	*488.21*	407.6
11	480.10	0.9	473.24	3.7	477.33	5.9	*459.80*	493.5
12	451.69	1.3	447.29	4.8	447.77	7.0	*432.33*	547.1
13	423.16	1.5	419.77	6.2	418.88	7.7	*406.13*	650.8
14	407.29	2.0	405.23	7.1	403.85	8.8	*391.27*	758.3
15	391.59	2.3	384.80	8.7	384.24	9.7	*372.63*	840.7
16	393.03	2.7	386.04	10.2	385.13	10.6	*379.33*	898.5
17	400.53	3.3	393.36	11.7	395.93	11.9	*380.23*	981.2
18	411.16	3.7	403.26	13.0	407.20	13.6	*398.27*	1,051.0
Average	478.44	1.7	474.34	6.0	475.48	7.6	*459.13*	636.4

In general, the proposed method consistently outperforms the other three methods in terms of the makespan of all four scenarios. Take Table 4 for scenario 1 as an example, the average makespan by using our proposed method is 458.66 seconds, which is better than the random method (476.4 seconds), the congestion-aware method (472 seconds), and the MD method (473.1 seconds). This demonstrates that our simulation-based scheduling method is capable of providing high-quality solutions to the robotic put wall, owing to the metaheuristic integrated with the derived system's ability to simulate various solutions in a short period. Table 4 shows that the congestion-aware method outperforms the MD method slightly. The choice of the shortest distance may not result in higher productivity than the congestion-aware method, because the shortest distance may cause congestion, lowering overall system productivity. Similar observations can also be found in Table 5–7.

Table 6 Compared computational results with respect to scenario 3

Settings	*Random*		*Congestion aware*		*MD*		*Our proposed*	
N_k	*makespan (s)*	*CT (ms)*	*makespan (s)*	*CT (ms)*	*makespan (s)*	*CT (ms)*	*makespan (s)*	*CT (ms)*
7	1,380.22	0.4	1,378.34	0.8	1,384.54	2.4	*1,324.82*	287.5
8	1,229.56	0.5	1,221.78	1.1	1,229.32	3.3	*1,185.70*	329.4
9	1,116.24	0.6	1,100.24	1.7	1,110.32	4.5	*1,074.05*	370.3
10	1,013.73	0.8	1,011.80	2.6	1,011.25	5.1	*975.82*	430.1
11	949.24	0.9	934.46	4.0	944.29	6.0	*910.24*	462.3
12	881.60	1.3	873.78	5.0	874.06	7.0	*843.78*	531.5
13	833.14	1.5	825.57	6.4	823.90	7.6	*798.74*	637.8
14	797.63	2.0	792.93	7.1	789.82	8.8	*765.70*	761.0
15	775.33	2.4	769.03	9.0	765.55	10.0	*738.17*	823.6
16	769.27	2.7	765.08	10.4	766.30	10.9	*737.46*	889.0
17	780.19	3.2	776.12	12.0	773.85	13.0	*743.88*	964.4
18	802.06	3.7	789.24	12.8	790.07	13.7	*770.18*	1,031.4
Average	944.02	1.7	936.53	6.1	938.61	7.7	*905.71*	626.5

Table 7 Compared computational results with respect to scenario 4

Settings	*Random*		*Congestion aware*		*MD*		*Our proposed*	
N_k	*makespan (s)*	*CT (ms)*	*makespan (s)*	*CT (ms)*	*makespan (s)*	*CT (ms)*	*makespan (s)*	*CT (ms)*
7	1,383.67	0.4	1,384.30	0.8	1,388.33	2.6	*1,330.77*	274.1
8	1,230.12	0.5	1,229.53	1.1	1,231.01	3.6	*1,187.31*	339.0
9	1,119.66	0.6	1,111.37	1.7	1,117.85	4.4	*1,074.87*	381.2
10	1,019.25	0.8	1,014.09	2.7	1,014.13	5.1	*977.44*	421.3
11	953.28	1.0	937.66	3.8	946.97	6.0	*911.57*	478.7
12	883.03	1.3	875.77	4.8	876.00	7.0	*845.37*	548.0
13	832.66	1.6	825.18	6.2	823.63	7.7	*800.13*	632.5
14	800.03	1.9	793.72	7.4	790.25	9.0	*766.03*	751.0
15	775.31	2.4	769.90	8.7	767.63	10.0	*738.31*	821.9
16	770.33	2.6	767.24	10.1	770.77	10.5	*737.92*	875.4
17	781.24	3.1	776.38	11.6	777.07	12.2	*745.34*	990.1
18	803.97	3.6	790.99	12.9	793.07	14.0	*775.33*	1,019.3
Average	946.05	1.7	939.68	6.0	941.39	7.7	*907.53*	627.7

The compared computation times of these methods for all four scenarios are also recorded in Tables 4–7. It should be noted that our proposed method takes significantly longer to compute than the other three methods. This is because for the decision process, multiple simulations are conducted, and each decision is evaluated after the entire simulation is completed. Nonetheless, the computation time is less than one second, satisfying the requirement of a real-time decision every second. The three rule-based methods (random, congestion-aware, and MD) all have short computation times, with the longest being less than 15 ms. These rules do produce quick results.

Figure 7 Fitness curves by different crossover and mutation probabilities (see online version for colours)

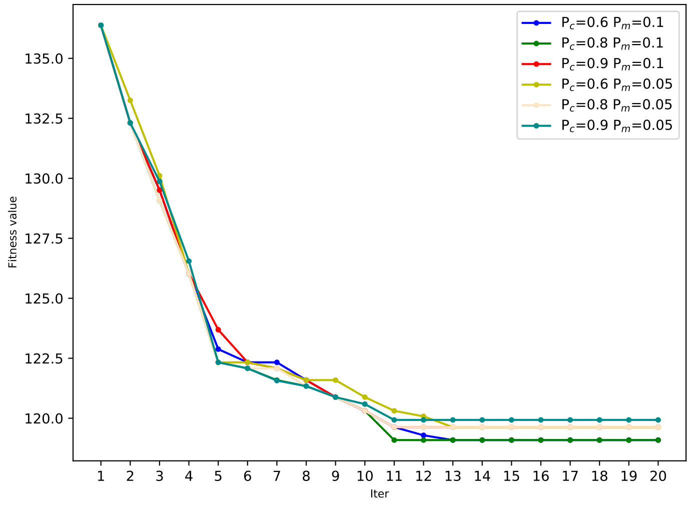
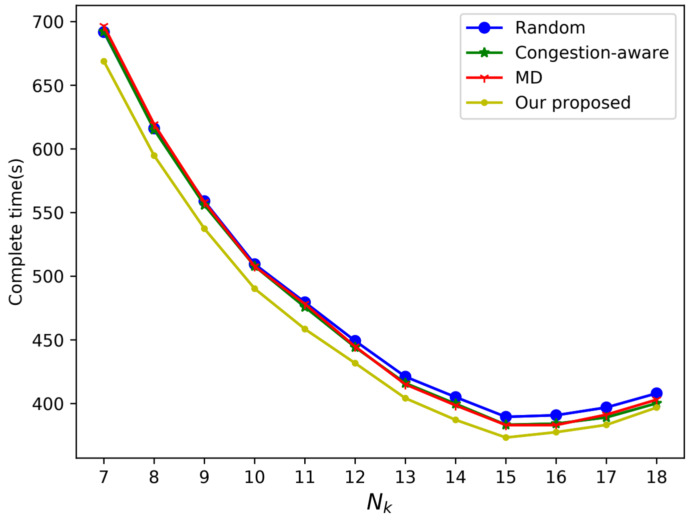


Figure 8 Influence of different N_k values on the makespan of four methods for scenario 1 (see online version for colours)



From Tables 4–7, the optimal number of RGVs N_k can be determined for each scenario, by examining the performance of the makespan as N_k changes (from 7 to 18). For instance, when $N_k = 15$, our scheduling method achieves the shortest makespan for scenarios 1–2. When N_k increases from 7 to 16, the makespan decreases, while it increases when N_k increases from 16 to 18. The tendency to decline at the beginning and rise up late can also be seen in the makespan of the other three methods for each scenario.

Figure 8 illustrates the effects of different N_k values on the makespan for scenario 1. This figure clearly shows the tendency to decline and rise up when N_k becomes larger, as well as the proposed method's advantage over the other three methods. For scenario 1, the optimal N_k is suggested to be 15, which determines the shortest time for all four methods.

Table 8 Compared results when varying the rescheduling interval for scenario 1

Interval	*100 ms*		*500 ms*		*1 s*		*5 s*	
N_k	*makespan (s)*	*CT (ms)*	*makespan (s)*	*CT (ms)*	*makespan (s)*	*CT (ms)*	*makespan (s)*	*CT (ms)*
12	428.48	551.3	430.24	564.4	431.72	569.9	435.26	556.9
13	401.06	641.1	402.62	646.4	404.20	646.4	407.68	643.2
14	383.66	744.9	385.71	754.2	387.18	764.1	391.09	753.8
15	369.91	814.9	371.78	827.9	373.34	816.2	377.00	812.7
16	374.62	880.4	375.40	883.9	377.55	880.2	380.31	876.4
17	380.27	964.7	381.61	963.3	383.27	953.8	386.75	960.7
Average	389.67	766.2	391.23	773.4	392.88	771.8	396.35	767.3

Table 5 compares the makespan and computation time when the rescheduling interval is changed to further investigate its effects on these two indicators for scenario 1. The interval is changed from 100 ms to 5 seconds for comparison. Table 5 shows that when N_k is changed from 12 to 17, the makespan decreases and then increases, which is consistent with the trend in Table 4. When N_k reaches 15, productivity reaches a limit, and when N_k exceeds 15, productivity begins to decline because more congestion is created. The computation time increases as N_k increases because more decision variables are included in the optimisation problem.

Table 5 shows that the makespan can be reduced as the rescheduling interval becomes shorter. In practice, however, this reduction is not possible because the computation time is significantly longer than the interval. 1 second is a good choice for achieving a balance between makespan and computation time. Because the optimisation problem is the same size regardless of the rescheduling interval value, the computation time does not vary significantly.

5 Conclusions and future research

In this paper, we present a novel dynamic scheduling method for a robotic put wall system to handle randomly arriving items. By combining simulation and metaheuristics at predefined intervals, the proposed scheduling method iteratively accesses the performance of dealing with arrival items in the put wall. This integration takes advantage of the simulation to model the complex process of the put wall system and the genetic algorithm to effectively optimise the system performance. The simulation is implemented by the state-of-the-art software Tecnomatix Plant, and the operation details of the process can be included in the simulation to better represent the real system. The genetic algorithm is customised to solve the combinatorial optimisation problem of determining a suitable destination. Using the proposed integrated approach, the minimal value of the vehicle number can also be obtained.

The benefits of the proposed methodology are demonstrated through case studies derived from real-world applications at a put wall manufacturer company. The proposed method is compared to three rule-based real-time scheduling methods, and its makespan outperforms these rule-based methods. A sufficiently good solution can be obtained in a reasonable amount of time by using a genetic algorithm designed for combinatorial optimisation problems. Because multiple simulations are performed to find the best fitness value of the genetic algorithm, our proposed method takes longer to compute than the three rule-based methods.

Regarding future research, the following aspects can be considered: first, the impact of vehicle speed, goods arrival interval, vehicle acceleration, and other parameters on system performance can be considered. Second, the structure of the picking system can be optimised, such as the overall operation logic of the picking system, the number of feeding ports, and so on. Third, combining plant simulation software with other optimisation algorithms is an option.

Acknowledgements

This research is supported in part by the National Natural Science Foundation of China under Grant 62173311 and 61703372, in part by the College Youth Backbone Teacher Project of Henan Province under Grant 2021GGJS001, in part by Henan Scientific and Technological Research Project under Grant 222102220123, and in part by the Training Project of Zhengzhou University under Grant JC21640030.

References

Ardjmand, E., Bajgiran, O.S. and Youssef, E. (2019) 'Using list-based simulated annealing and genetic algorithm for order batching and picker routing in put wall based picking systems', *Applied Soft Computing*, Vol. 75, pp.106–119.

Ardjmand, E., Youssef, E.M., Moyer, A., Ii, W.A.Y., Weckman, G.R. and Shakeri, H. (2020) 'A multi-objective model for minimising makespan and total travel time in put wall-based picking systems', *International Journal of Logistics Systems and Management*, Vol. 36, No. 1, pp.138–176.

Azadeh, K., De Koster, R. and Roy, D. (2019) 'Robotized and automated warehouse systems: review and recent developments', *Transportation Science*, Vol. 53, No. 4, pp.917–945.

Bangsow. S. (2020) *Tecnomatix Plant Simulation: Modeling and Programming by Means of Examples*, Springer, Cham, Switzerland.

Bódis, T. and Botzheim, J. (2018) 'Bacterial memetic algorithms for order picking routing problem with loading constraints', *Expert Systems with Applications*, Vol. 105, pp.196–220.

Boysen, N., de Koster, R. and Füßler, D. (2021) 'The forgotten sons: warehousing systems for brick-and-mortar retail chains', *European Journal of Operational Research*, Vol. 288, No. 2, pp.361–381.

Boysen, N., de Koster, R. and Weidinger, F. (2019) ‘Warehousing in the e-commerce era: a survey’, *European Journal of Operational Research*, Vol. 277, No. 2, pp.396–411.

Boysen, N., Stephan, K. and Weidinger, F. (2019) ‘Manual order consolidation with put walls: the batched order bin sequencing problem’, *EURO Journal on Transportation and Logistics*, Vol. 8, No. 2, pp.169–193.

Chabot, T., Lahyani, R. and Coelho, L.C. and Renaud, J. (2017) ‘Order picking problems under weight, fragility and category constraints’, *International Journal of Production Research*, Vol. 55, No. 21, pp.6361–6379.

Cui X.Q. (2020) ‘Multi-objective flexible flow shop batch scheduling problem with renewable energy’, *International Journal of Automation and Control*, Vol. 25, Nos. 5–6, pp.519–553.

De Koster, R., Le-Duc, T. and Roodbergen, K.J. (2007) ‘Design and control of warehouse order picking: a literature review’, *European Journal of Operational Research*, Vol. 182, No. 2, pp.481–501.

Fabri, M., Ramalhinho, H., Oliver, M. and Muñoz, J.C. (2022) ‘Internal logistics flow simulation: a case study in automotive industry’, *Journal of Simulation*, Vol. 16, No. 2, pp.204–216.

Gil-Borrás, S., Pardo, E.G., Alonso-Ayuso, A. and Duarte, A. (2021) ‘A heuristic approach for the online order batching problem with multiple pickers’, *Computers & Industrial Engineering*, Vol. 160, p.107517.

Henn, S. (2015) ‘Order batching and sequencing for the minimization of the total tardiness in picker-to-part warehouses’, *Flexible Services and Manufacturing Journal*, Vol. 27, No. 1, pp.86–114.

Henn, S. and Schmid, V. (2013) ‘Metaheuristics for order batching and sequencing in manual order picking systems’, *Computers & Industrial Engineering*, Vol. 66, No. 2, pp.338–351.

Invata Automated Robotic Put Wall Solutions (2023) [online] https://www.invata.com/automated-put-wall-solutions/ (accessed 1 June 2023).

Kramer, O. (2017) *Genetic Algorithm Essentials*, Springer, Berlin, Germany.

Laha, D. and Chakraborty, U.K. (2010) ‘Minimising total flow time in permutation flow shop scheduling using a simulated annealing-based approach’, *International Journal of Automation and Control*, Vol. 4, No. 4, pp.359–379.

Lamballais, T., Roy, D. and De Koster, M. (2017) ‘Estimating performance in a robotic mobile fulfillment system’, *European Journal of Operational Research*, Vol. 256, No. 3, pp.976–990.

Lee, H-Y. and Murray, C.C. (2019) ‘Robotics in order picking: evaluating warehouse layouts for pick, place, and transport vehicle routing systems’, *International Journal of Production Research*, Vol. 57, No. 18, pp.5821–5841.

Lee, I.G., Chung, S.H. and Yoon, S.W. (2020) ‘Two-stage storage assignment to minimize travel time and congestion for warehouse order picking operations’, *Computers & Industrial Engineering*, Vol. 139, p.106129.

Luo, L., Zhao, N., Zhu, Y. and Sun, Y. (2023) ‘A^* guiding DQN algorithm for automated guided vehicle pathfinding problem of robotic mobile fulfillment systems’, *Computers & Industrial Engineering*, Vol. 178, p.109112.

Masae, M., Glock, C.H. and Vichitkunakorn, P. (2021) ‘A method for efficiently routing order pickers in the leaf warehouse’, *International Journal of Production Economics*, Vol. 234, p.108069.

Matusiak, M. and De Koster, R., Kroon, L. and Saarinen, J. (2014) ‘A fast simulated annealing method for batching precedence-constrained customer orders in a warehouse’, *European Journal of Operational Research*, Vol. 236, No. 3, pp.968–977.

Öncan, T. (2015) ‘Milp formulations and an iterated local search algorithm with tabu thresholding for the order batching problem’, *European Journal of Operational Research*, Vol. 243, No. 1, pp.142–155.

Sun, Y., Zhao, N. and Lodewijks, G. (2021) 'An autonomous vehicle interference-free scheduling approach on bidirectional paths in a robotic mobile fulfillment system', *Expert Systems with Applications*, Vol. 178, p.114932.

Tan, Z., Li, H. and He, X. (2021) 'Optimizing parcel sorting process of vertical sorting system in e-commerce warehouse', *Advanced Engineering Informatics*, Vol. 48, p.101279.

Van Gils, T., Ramaekers, K., Caris, A. and De Koster, R.B. (2018) 'Designing efficient order picking systems by combining planning problems: state-of-the-art classification and review', *European Journal of Operational Research*, Vol. 267, No. 1, pp.1–15.

Van Vianen, T., Ottjes, J. and Lodewijks, G. (2016) 'Belt conveyor network design using simulation', *Journal of Simulation*, Vol. 10, No. 3, pp.157–165.

Wagner, S. and Mönch, L. (2022) 'A variable neighborhood search approach to solve the order batching problem with heterogeneous pick devices', *European Journal of Operational Research*, Vol. 304, No. 2, pp.461–475.

Xin, J., Meng C., D'Ariano, A., Schulte, F., Peng J. and Negenborn R. (2023) 'Energy-efficient routing of a multirobot station: a flexible time-space network approach', *IEEE Transactions on Automation Science and Engineering*, Vol. 20, No. 3, pp.2022–2036.

Xin, J., Negenborn R. and Lodewijks G. (2014) 'Rescheduling of interacting machines in automated container terminals', *IFAC Proceedings Volumes*, Vol. 47, No. 3, pp.1698–1704.

Xin, J., Yu, B., D'Ariano, A., Wang, H. and Wang, M. (2022) 'Time-dependent rural postman problem: time-space network formulation and genetic algorithm', *Operational Research*, Vol. 22, No. 3, pp.2943–2972.

Yu, M. and De Koster, R.B. (2009) 'The impact of order batching and picking area zoning on order picking system performance', *European Journal of Operational Research*, Vol. 198, No. 2, pp.480–490.