Hindawi
Journal of Sensors
Volume 2022, Article ID 2056617, 9 pages
https://doi.org/10.1155/2022/2056617

Research Article

Multirobot Adaptive Task Allocation of Intelligent Warehouse Based on Evolutionary Strategy

Yifan Liu,[1] Fei Liu,[1] Li Tang,[2] Chuanzheng Bai,[1] and Li Liu[1]

[1]*School of Information and Electrical Engineering, Ludong University, Yantai 264025, China*
[2]*School of Physics and Optoelectronic Engineering, Ludong University, Yantai 264025, China*

Correspondence should be addressed to Fei Liu; liufeildu@163.com

Received 26 July 2021; Accepted 14 March 2022; Published 18 April 2022

Academic Editor: Yunze He

To solve the dynamic and real-time problem of multirobot task allocation in intelligent warehouse system under parts-to-picker mode, this paper presents a combined solution based on adaptive task pool strategy and Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) algorithm. In the first stage of the solution, a variable task pool is used to store dynamically added tasks, which can dynamically divide continuous and large-scale task allocation problems into small-scale subproblems to solve them to meet dynamic requirements. And an adaptive control strategy is used to automatically adjust the total number of tasks in the task pool to achieve a trade-off among throughput, energy consumption, and waiting time, which has better adaptability than manually adjusting the size of the task pool. In the second stage of the solution, when the task pool is full, tasks in the task pool will be assigned to robots. For the task allocation problem, this paper regards it as an optimization problem and uses the CMA-ES algorithm to find the optimal task assignment solution for all the robots. By comparing with fixed threshold method under 56 different task pool sizes, the experimental results show that the throughput can be close to reaching the optimal level, and the average distance traveled by robots to handle each unit is lower using adaptive threshold method; so, adaptive task pool solution has better adaptability and can find the optimal task pool size by itself. This method can satisfy the dynamic and real-time requirements and can be effectively applied to the intelligent warehouse system.

1. Introduction

In recent years, the orders of various e-commerce platforms have soared, and the scale of distribution centers has become increasingly large, which has brought great challenges to the traditional logistics industry [1]. In the traditional warehouse, 60% to 70% of the workers' time is spent on picking up goods [2], and the efficiency is extremely low. Therefore, more and more automatic machines and equipment have been applied in the field of warehouse [2]. Many companies have started to adopt a new kind of parts-to-picker intelligent warehouse system, such as Kiva system [3]. In the system as shown in Figure 1, robots transport the shelves from storage areas to workstations, and workers need to wait at the stations. When the shelves reach the workstations, they take the needed goods from the shelves or store bundles into the shelves. It has been proved that this kind of the intelligent warehouse system greatly saves labor cost and improves the efficiency of warehouse operation [4].

Cooperative control of multiple mobile robots is the key to realize intelligent warehousing. In a warehouse as shown in Figure 1, there are often numerous tasks such as replenishment and picking, as well as numerous robots to perform these tasks. In addition, the costs of different robots to perform a task are also different. Therefore, the efficiency of the warehouse is determined by selecting suitable robots to perform specific tasks. This is a typical multirobot task allocation (MRTA) problem [5]. With the operation of the warehouse, tasks and the warehouse environment will constantly change. How to find a better task allocation scheme for pick-task and replenishment-task assignment in such a highly dynamic environment [3, 4] is the focus of this paper.

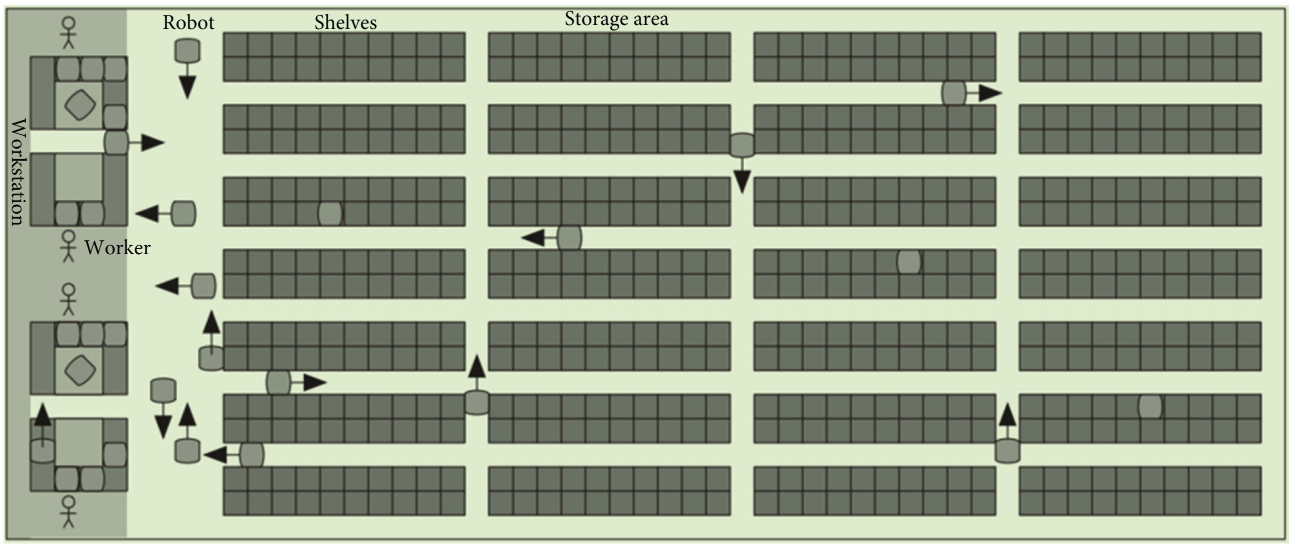

Figure 1: Parts-to-picker intelligent warehouse system from ref. [4].

MRTA is one of the most challenging problems in the multirobot system [6]. Market-based methods are the most studied methods at present, such as the single-task auction algorithm proposed in ref. [7]. In order to solve the problem that the single-task auction algorithm is difficult to get the optimal solution, a combined auction algorithm which considers the correlation between tasks was proposed in ref. [8]. When the number of robots and tasks is small, MRTA can be regarded as a zero-one integer linear programming problem and solved by simplex method, branch and bound method, Hungarian algorithm [9], etc. For example, the Hungarian algorithm was adopted in ref. [10] to solve the role assignment problem in robot soccer game. There are also some thresholding based methods such as ALLIANCE [11] and Broadcast of Local Eligibility (BLE) [12], which have good real-time, fault tolerance, and robustness, but usually only local optimal solution can be obtained. For large-scale problems, the heuristic algorithm can effectively reduce solution space and improve search efficiency. For example, in ref. [13], the heuristic algorithm was adopted to solve the task assignment problem in multi-core processor. Evolutionary algorithms are mature global optimization methods with high robustness and wide applicability, which can effectively deal with complex problems that are difficult to be solved by traditional optimization algorithms. Various evolutionary algorithms such as genetic algorithm and simulated annealing algorithm have been widely used in MRTA problem. In ref. [14], the genetic algorithm was used to solve the time-extended multirobot task allocation problem in the case of disaster. A hybrid genetic and ant colony algorithm was proposed in ref. [15] to improve the solving accuracy of the genetic algorithm. In ref. [16], the genetic algorithm was used to solve MRTA problem in the intelligent warehouse. Ref. [17] designed an improved quantum evolutionary algorithm based on the niche coevolution strategy and enhanced particle swarm optimization (IPOQEA) to solve the airport gate allocation problem. In ref. [18], an improved quantum-inspired cooperative coevolution algorithm with multistrategy is used to solve the knapsack problem and the actual airport gate allocation problem. Refs. [17–20] use the cooperative coevolution framework to divide the complex optimization problem into several subproblems, and these subproblems were solved by independent searching in order to improve the solution efficiency. Similarly, the situation where the number of tasks is variable in an intelligent warehouse can be studied using the idea of divide-and-conquer in Refs. [17–20].

Therefore, we use a task pool to store dynamically added tasks and propose an adaptive control strategy to automatically adjust the task pool size according to the current environment. When the task pool is full, the tasks in the pool will be assigned to the robots. Then, the task allocation problem is regarded as an optimization problem and solved by the CMA-ES algorithm [21].

2. Problem Formulation

The intelligent warehouse system consists of many movable shelves and robots as well as some workstations. The robots transport the needed shelves from the storage area to the workstations, and the workers can complete the replenishment and picking without moving. A typical intelligent warehouse layout (a screenshot from the open source software RAWSim-O [22]) is shown in Figure 2. In the figure, the four squares on the left represent the replenishment station, and the replenished bundles are temporarily stored here waiting for shelves. The four squares on the right represent picking stations. After receiving orders, the system will use a special algorithm to assign orders to different stations. There will be an upper limit on the number of orders in the stations [23]. The squares in the middle area are the shelves, in which the goods in the warehouse are stored. Shelves can be lifted and moved by robots. The circles in the figure are robots. A robot can carry a shelf to move. When a robot does not carry a shelf, it can move freely under the shelf.

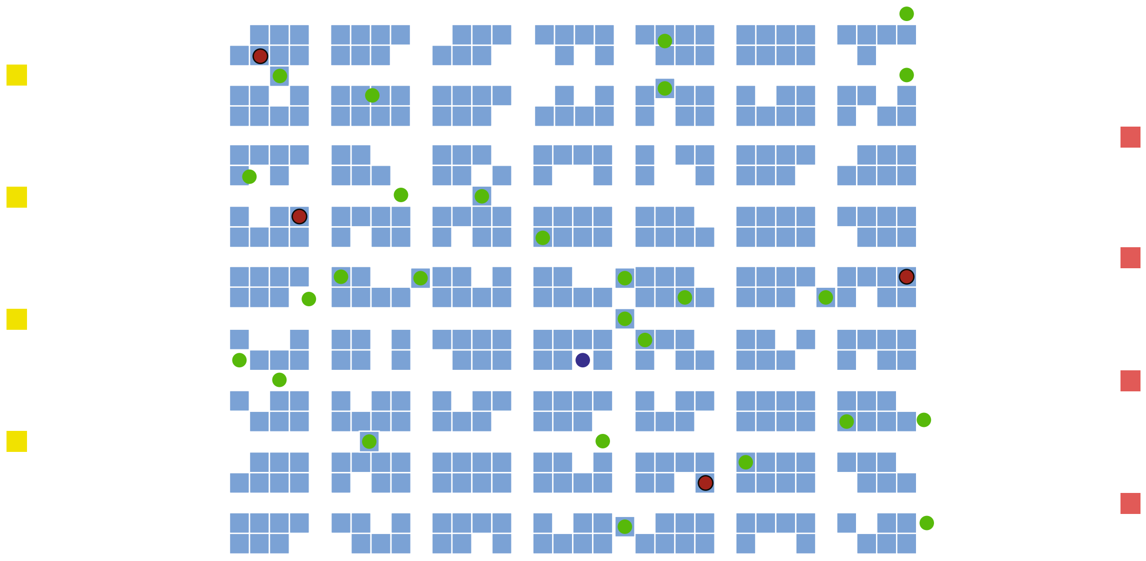

Figure 2: A typical intelligent warehouse layout from ref. [22].

In order to facilitate problem analysis, we make the following assumptions:

(1) Robots are all isomorphic and travel at exactly the same speed. They can only move forward, backward, left, and right.

(2) The time for a robot to lift a shelf and stay at a workstation is very short, which can be ignored.

(3) Every robot carries the required shelf and travels from the position of the shelf to the designated station and then carries the shelf back to its original location.

The shelf selection algorithm will select shelves for each workstation according to requirements. The selected shelves need to be transported from the shelf storage area to the appropriate station for picking up or replenishing goods, and then they are transported back to the original position, which is the task of the robots. If a robot is not assigned a task, it will move to a special resting area for rest. How to reasonably assign tasks to robots is the problem to be studied in this paper.

Referring to ref. [16], suppose that there are m tasks (refers to all tasks from the beginning to the end of the warehouse operation) and n robots in the warehouse, the set of tasks is $T=\{t_1, t_2, t_3,\cdots,t_m\}$, and the set of robots is $R=\{r_1, r_2, r_3\cdots,r_n\}$. The set of tasks assigned to robot r_i is T_i, which is a subset of T. $T_1 \cup T_2 \cup T_3 \cup \cdots \cup T_n = T$ and $T_1 \cap T_2 \cap T_3 \cap \cdots \cap T_n = \varnothing$. Let $T_i=\{t_{i1}, t_{i2}, t_{i3},\cdots,t_{ik}\}$ and T_i is ordered, and then the sequence of tasks to be completed by the robot r_i is $t_{i1} \longrightarrow t_{i2} \longrightarrow t_{i3} \longrightarrow \cdots \longrightarrow t_{ik}$. The cost of robot r to complete its task sequence can be expressed as

$$C(r_i) = I(r_i, t_{i1}) + \sum_{h=1}^{k} S(t_h) + \sum_{h=1}^{k-1} R(t_h, t_{h+1}), \tag{1}$$

where $C(r_i)$ represents the cost of the robot r_i to complete all tasks. Since all robots travel at the same speed, the cost can be expressed as the distance traveled by the robot. The robot can only move forward, backward, left, and right; so, the distance traveled between the two points can be expressed as Manhattan distance.

$I(r_i, t_{i1})$ represents the cost for the robot to get from the initial position to the position of required shelf for the first task t_{i1}. Let the initial coordinate of the robot be (x_r, y_r) and the coordinate of the required shelf for the first task be (x_{t1}, y_{t1}), and then

$$I(r_i, t_{i1}) = |x_r - x_{t1}| + |y_r - y_{t1}|. \tag{2}$$

$S(t_h)$ represents the cost for the robot to complete task t_h , which is only related to task t_h itself. It can be represented by the distance that after the robot carries the required shelf, it travels from the position of the required shelf for the task to the designated station and then returns to the shelf's original position from the station. Let the coordinate of required shelf for task t_h be (x_p, y_p) and the coordinate of target station be (x_s, y_s), and then

$$S(t_h) = \left(\left|x_p - x_s\right| + \left|y_p - y_s\right|\right) * 2. \tag{3}$$

$R(t_h, t_{h+1})$ represents the cost for the robot to reach the starting position of the next task t_{h+1} after completing task t_h. Since the robot needs to transport the shelf back to the original position after completing task t_h, it can be directly represented by the Manhattan distance from the position of required shelf for task t_h to the position of required shelf for task t_{h+1}. Let the coordinate of required shelf for task t_h be (x_{p1}, y_{p1}) and the coordinate of required shelf for task t_{h+1} be (x_{p2}, y_{p2}), and then

$$R(t_h, t_{h+1}) = \left|x_{p1} - x_{p2}\right| + \left|y_{p1} - y_{p2}\right|. \tag{4}$$

In order to make the overall allocation scheme as optimal as possible, we consider the following two optimization objectives:

(1) The maximum time taken by all robots to complete all tasks (C_{time})

(2) The mean distance traveled by all robots (C_{distance})

where

$$C_{\text{time}} = \max_i C(r_i),$$
$$C_{\text{distance}} = \frac{\sum_{i=1}^{n} C(r_i)}{n}. \tag{5}$$

C_{time} describes the efficiency of the robots to complete tasks. The smaller C_{time} is, the less time the robots take to complete all tasks, and the higher the efficiency is. C_{distance} describes the power consumption of the multirobot system. The smaller C_{distance} is, the shorter the total travel distance of all robots is, and the lower the power consumption is. The goal of the method studied in this paper is to reasonably assign all tasks in the system to all robots so that these two values can be as small as possible.

3. Method

3.1. Architecture. With the entry of new orders, new tasks are constantly generated and must be completed as soon as possible; so, the warehouse system is a highly dynamic and real-time system. In such a highly dynamic system, it is difficult to find the global optimal solution; so, the problem is divided into many subproblems. Specifically, we created a task pool P. When a new task is generated, it is immediately added to P. When the number of tasks in the task pool P reaches the threshold value (automatic adjustment of the threshold will be described in Section 3.3), the CMA-ES method in Section 3.2 is used to allocate the tasks in the task pool to robots. The robots insert the new task sequence allocated into the rear of the previous unfinished task sequence, and then the task pool is emptied. The robots execute tasks according to their own task sequence, and the executed tasks are deleted from the sequence. As the new tasks are generated again, the tasks are added to P again. Loop until the warehouse stops running. In Figure 3, the specific steps are as follows:

Step 1. Initialize the task pool size and set the task pool P to be empty. For all robots, initialize task sequence T_i of every robot r_i.

Step 2. The threshold of the task pool size is automatically adjusted using adaptive control strategy in Section 3.3.

Step 3. New tasks are constantly added to P. Jump to step 4 when the number of tasks in the task pool reaches the threshold.

Step 4. The tasks in the task pool are assigned to the robots using the CMA-ES method in Section 3.2, and for all robots, the new task sequence assigned to robot r_i is inserted at the end of the current task sequence T_i.

Step 5. Clear the task pool P and jump to step 2.

The above solution in Figure 3 is executed by the central controller, and the robot only needs to execute the tasks according to the assigned task sequence. The parallel operation of the two parts enables the robots to be busy all the time, which saves time and meets the requirement of real-time storage system.

3.2. CMA-ES Algorithm. As mentioned in Section 3.1, tasks are assigned to robots when the number of tasks in the task pool reaches the threshold. This problem is regarded as an optimization problem in a static environment. This is a NP-hard problem, and the CMA-ES algorithm is used to find the optimal solution. The successful application in many fields [24–26] proves that the CMA-ES algorithm is a good search algorithm.

3.2.1. Representation of Solutions. Referring to ref. [27], for the task allocation problem with m tasks and n robots, a candidate to represent a task assignment scheme is $X = [x_1, x_2, x_3 \cdots x_m]$. X contains m real numbers, and for each real number x_i, it satisfies $1 \leq x_i < n + 1, i = 1, 2, 3, \cdots, m$, where x_i means task i is performed by robot $\text{Int}(x_i)$, and $\text{Int}(x_i)$ means the integer of real number x_i. If $\text{Int}(x_i) = \text{Int}(x_j), i \neq j$, this means that the task x_i and x_j are both assigned to the same robot, and the task represented by the smaller number between x_i and x_j is executed first. If $x_i = x_j$, the execution order of these two tasks is determined randomly.

For example, there are 8 tasks (represented by numbers 1, 2, 3,..., 8) and 3 robots (represented by numbers 1, 2, 3), and an individual [1.7, 3.8, 2.2, 1.3, 2.8, 1.5, 3.3, 3.7] is generated. Then, the task sequence assigned to robot 1 is $4 \longrightarrow 6 \longrightarrow 1$. The task sequence assigned to robot 2 is $3 \longrightarrow 5$. The task sequence assigned to robot 3 is $7 \longrightarrow 8 \longrightarrow 2$.

3.2.2. Fitness Function. Fitness function is used to evaluate candidates. For the CMA-ES algorithm, individuals with lower fitness value are more excellent. In Section 2, two optimization goals are proposed for the whole system: one is the time C_{time} for the robots to complete all tasks; the second is the mean driving distance C_{distance} of all robots. Each planning can be regarded as a subproblem of the whole. For each subproblem, in order to achieve the optimal overall performance, these two goals are still considered; so, fitness function f is calculated through the following equation [16]:

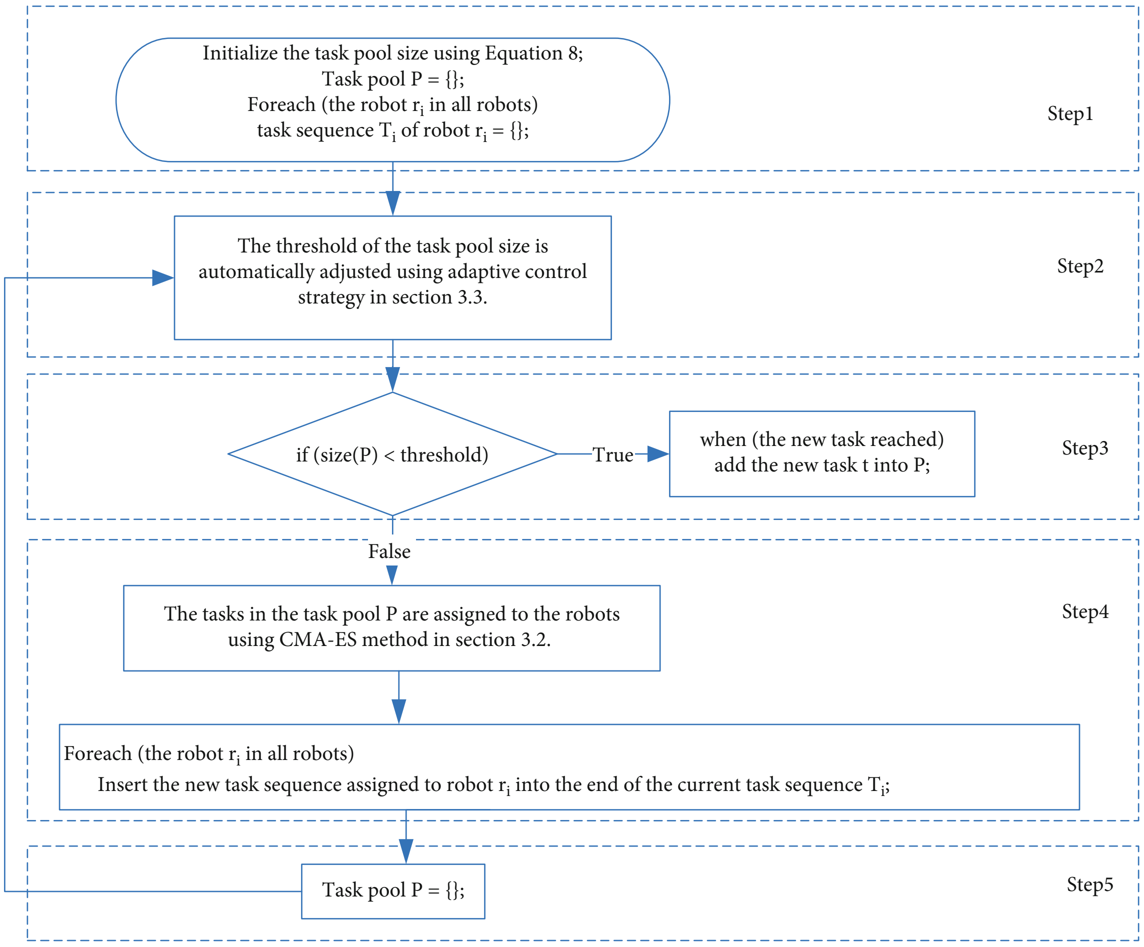
Initialize the task pool size using Equation 8;
Task pool P = {};
Foreach (the robot r_i in all robots)
task sequence T_i of robot r_i = {};
Step1
The threshold of the task pool size is automatically adjusted using adaptive control strategy in section 3.3.
Step2
if (size(P) < threshold)
True
when (the new task reached) add the new task t into P;
Step3
False
The tasks in the task pool P are assigned to the robots using CMA-ES method in section 3.2.
Step4
Foreach (the robot r_i in all robots)
Insert the new task sequence assigned to robot r_i into the end of the current task sequence T_i;
Task pool P = {};
Step5

FIGURE 3: The flow chart of the combined solution based on adaptive task pool strategy and CMA-ES.

$$f = \alpha C'_{\text{time}} + (1-\alpha) C'_{\text{distance}}, 0 \le \alpha \le 1,$$
$$C'_{\text{time}} = \max_i C'(r_i), \tag{6}$$
$$C'_{\text{distance}} = \frac{\sum_{i=1}^{n} C'(r_i)}{n},$$

where α is a constant that can be adjusted according to the actual demand. If more attention is paid to the completion time of a single order, α can be increased. If more attention is paid to the energy consumption of all robots, α can be reduced. $C'(r_i)$ is the cost of robot r_i to execute the tasks in the current task sequence first and then execute the tasks according to the candidate. C'_{time} is the maximum time taken by the robots. C'_{distance} is the mean distance traveled by all robots. In the current moment, there may be unfinished tasks in the task sequence. The robot must first complete these tasks before performing the tasks assigned at the current moment. Therefore, for $C'(r_i)$, we divide it into two parts to calculate:

$$C'(r_i) = C'_1(r_i) + C'_2(r_i), \tag{7}$$

where $C'_1(r_i)$ is the cost for the robot to complete the tasks in the current task sequence, and $C'_2(r_i)$ is the cost for the robot to execute the tasks according to the candidate. $C'_1(r_i)$ and $C'_2(r_i)$ are represented by the distance traveled by the robot and calculated using the method described in Equation (1).

With this fitness function, we try to find the optimal solution at that moment in each optimization and try to approximate the global optimal solution by this method.

3.3. Automatic Adjustment of Task Pool. When the number of tasks in the task pool reaches the threshold, the tasks in the task pool will be assigned to the robots. The threshold plays a decisive role in the efficiency of assignment. The larger the threshold is, the more tasks will be involved in the optimization, and then the more the planned scheme will be close to the global optimal solution. If an optimization contains all the tasks in the system, the optimal solution found by the optimization will be the optimal solution of the whole system. But orders in the warehouse are added dynamically over time, so tasks are also generated dynamically. As the threshold increases, the time required for the task pool to be filled will also increase, and this situation will occur: the robot has finished all the tasks assigned to it, but the number of tasks in the task pool has not reached the threshold; so, the next optimization cannot start, and the robot can only wait. This leads to a waste of time and cannot meet the real-time of the warehouse system. Moreover,

```
**Input:** lastAdjustTime, currentTime, lastTasksCompleted, tasksCompleted, oldThreshold, lastAction
**Output:** newThreshold, lastAction
1: if $currentTime - lastAdjustTime > I$ then
2:   if $tasksCompleted = 0$ then
3:     $newThreshold \longleftarrow oldThreshold/2$
4:     $lastAction \longleftarrow -1$
5:   else if $tasksCompleted - lastTaskCompleted \geq 0$ then
6:     $newThreshold \longleftarrow newThreshold + lastAction$
7:   else
8:     $newThreshold \longleftarrow newThreshold - lastAction$
9:     $lastAction \longleftarrow -lastAction$
10: else
11:   $newThreshold \longleftarrow oldThreshold$
12: return newThreshold, lastAction
```

Algorithm 1: Adaptive control strategy.

because each workstation has an order capacity limit, there is also an upper limit on the total number of tasks in the system, and if the task pool size exceeds this upper limit, the number of tasks in the task pool will never reach the threshold, and the system will be stagnant. Therefore, it is very important to set a threshold of appropriate size.

Obviously, for different warehouses, the threshold should be set differently depending on the actual situation. Even for the same warehouse, the number of robots may be adjusted, and the rate of order generation may vary at different times; so, it is not appropriate to set the threshold to a fixed value. Therefore, we design an adaptive control strategy to dynamically adjust the task pool, as shown in Algorithm 1.

First, the setting of the initial threshold is important, which determines the speed of finding the optimal threshold. We believe that the size of the initial threshold should be related to the number of robots and the upper limit number of tasks in the warehouse. The upper limit number of tasks in the warehouse is related to the number of workstations and the capacity of each workstation. So, we propose the following heuristic formula to calculate the initial threshold:

$$\text{initialThreshold} = \frac{(\gamma * \text{stations} + \text{robots})}{2}, \tag{8}$$

where γ is a constant representing the average number of tasks per workstation in unit time, which is set according to the actual situation. stations is the number of stations, and robots is the number of robots. We set a time interval I (It is a constant that can be set according to actual requirements), and every I seconds, the threshold is adjusted (line 1). lastAction is used to record the last adjustment. We counted the total number of tasks completed by the robot from the last adjusted moment to the current moment, and the total number of tasks completed from the penultimate adjusted moment to the last adjusted moment, expressed by tasksCompleted and lastTasksCompleted, respectively. If taskCompleted is 0, indicating that the threshold has been set so high that the number of tasks has not reached the threshold, then simply cut the threshold in half and set lastAction to −1 (line 2, line 3, and line 4). If tasksCompleted is greater than or equal to lastTasksCompleted, it indicates that the last adjustment has had a positive effect on the system, and the same adjustment will be performed (line 5 and line 6). If tasksCompleted is less than lastTasksCompleted, it indicates that the last adjustment had a negative effect on the system, and the reverse adjustment will be performed (line 7 and line 8). In addition, lastAction will be reversed (line 9).

4. Experiments

We used RAWSim-O [22], an open source framework developed by Merschformann et al., as the experimental platform. RAWSim-O is a simulation framework that simulates the operation of an intelligent warehouse system and allows us to test our own methods.

We used the warehouse layout shown in Figure 2. In the warehouse layout, there are 32 robots and 550 shelves. The storage positions of the shelves are at the middle area of the layout. And there are four replenishment stations on the left and four picking stations on the right. To simplify the problem, we set the duration of a robot staying at a workstation to a very small value of 0.1.

For the assessment of performance we take the sum of SKUs (stock keeping unit) in both item bundles stored at the replenishment stations and orders picked at the picking stations as handled units. This represents the throughput of the warehouse, and the higher the better. We also look at the average distance traveled by robots to handle each unit. This can represent the power consumption of the multirobot system.

In order to test the impact of task pool threshold size on the allocation effect, we did 56 experiments, each experiment corresponding to different pool sizes. Each experiment was simulated for 24 hours with 10 repetitions.

Under different task pool sizes, the number of units handled by robots is shown in the blue solid line in Figure 4, and the average distance traveled by robots to handle each unit is shown in the blue solid line in Figure 5. The comparison results among different fixed threshold on handled units and travel distance per unit are shown in Table 1. The maximum number of handled units is 207583 when the fixed threshold is set to 18. The minimum number of travel

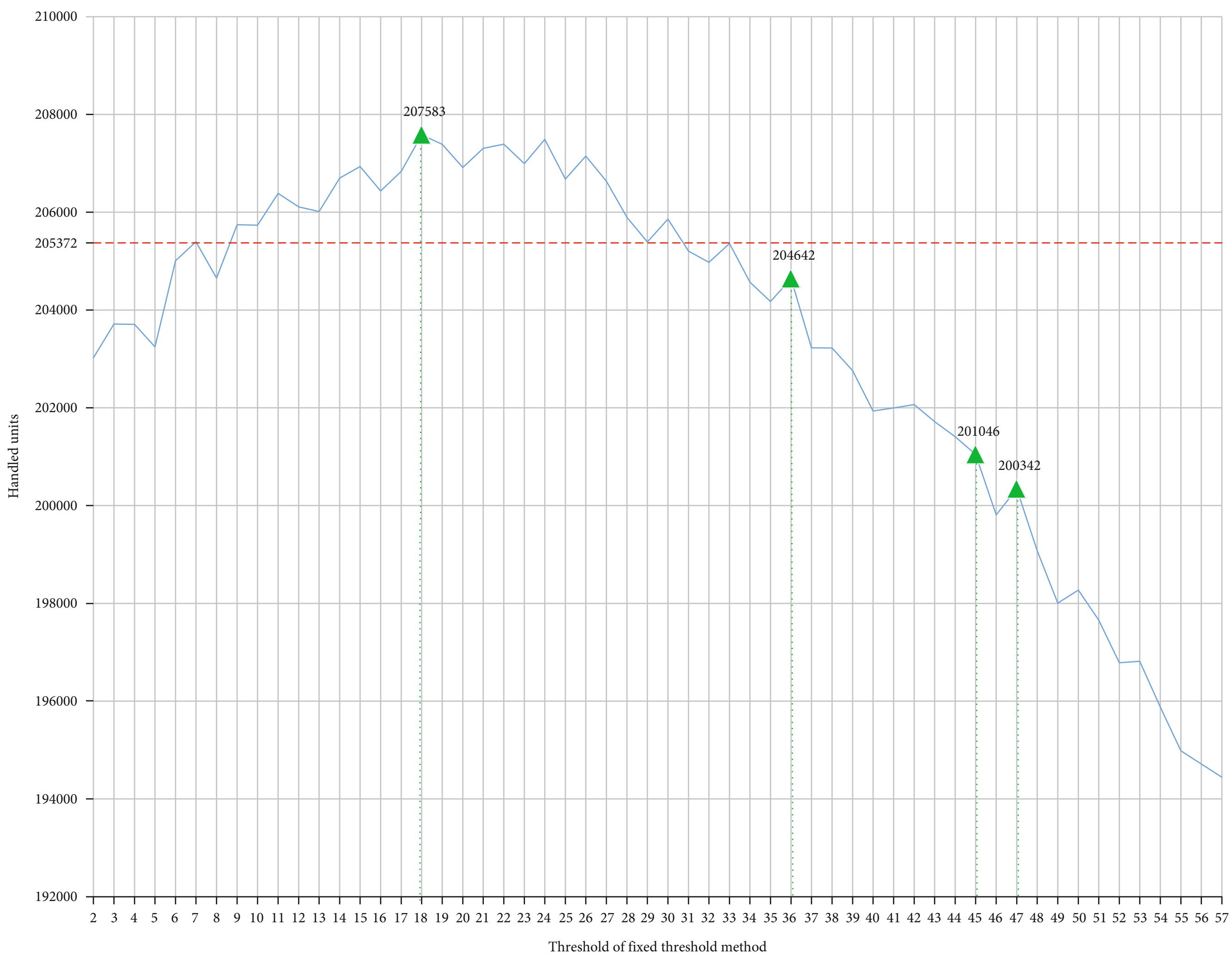

Figure 4: Comparison between adaptive threshold method and fixed threshold method on handled units. The red dotted line is the adaptive threshold method, and the blue solid line is the fixed threshold method.

distance per unit is 10.73 when the fixed threshold is set to 36, 45, or 47. According to Figures 4 and 5 and Table 1, it is not good to set the threshold too large or too small, which is consistent with our conjecture. If the threshold is set too small, the solution will be too far away from the global optimal solution; therefore, the number of handled units is small, and the travel distance per unit is large. If the threshold is set too large, the solution will be closer to the global optimal solution; so, the travel distance per unit is small, but the robot will have a long waiting time; therefore, the number of handled units will be small.

To sum up, a bad threshold can be very inefficient; so, setting the threshold manually is very risky. Therefore, a method of automatically adjusting threshold is necessary. We used the adaptive control strategy proposed by ourselves to conduct the experiment again, and all conditions were identical except the threshold. According to the workstation capacity, γ in Equation (8) was set to 4; so, the initial threshold was calculated as 32. The results are shown in Table 1. We compared the results with the fixed threshold approach, as shown in Figures 4 and 5. The red dotted line is the adaptive threshold method, and the blue solid line is the fixed threshold method. Compared with fixed threshold 18, the adaptive threshold method gets worse result in handled units but better result in travel distance per unit. Compared with fixed threshold 36, 45, and 47, the adaptive threshold method gets better result in handled units but worse result in travel distance per unit. Taken together, it can be seen from the two figures that the adaptive threshold method can be close to reaching the level when the threshold is set to the optimal in both indexes. The experimental results show that the proposed adaptive control strategy has good application effect.

5. Conclusion

In order to solve the dynamic and real-time problem of multirobot task allocation in the intelligent warehouse system, a combined solution based on adaptive task pool strategy and CMA-ES algorithm is proposed in the paper. In the early

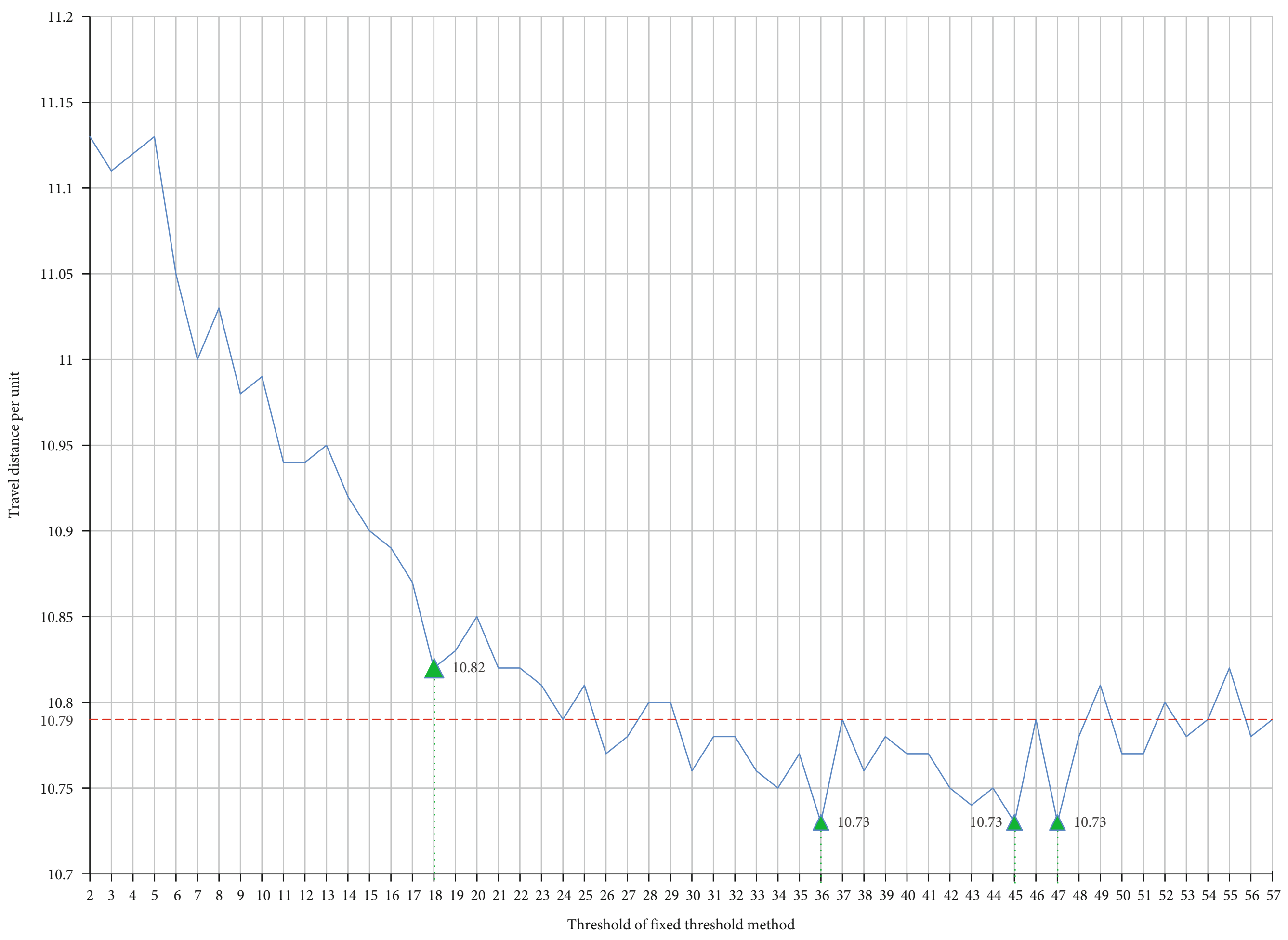

Figure 5: Comparison between adaptive threshold method and fixed threshold method on travel distance per unit. The red dotted line is the adaptive threshold method, and the blue solid line is the fixed threshold method.

Table 1: Comparison between adaptive threshold method and fixed threshold method on handled units and travel distance per unit.

Method (initial threshold)	Handled units	Travel distance per unit
Fixed threshold (18)	207583	10.82
Fixed threshold (36)	204642	10.73
Fixed threshold (45)	201046	10.73
Fixed threshold (47)	200342	10.73
Adaptive threshold (32)	205372	10.79

stage of the solution, the divide-to-conquer idea is used to design a variable task pool that is used to store dynamically added tasks. The variable task pool is designed to dynamically divide continuous and large-scale task allocation problems into small-scale subproblems to solve them to meet dynamic requirements. And an adaptive control strategy is used to automatically adjust the threshold of the task pool size in real time to achieve a trade-off among throughtput, energy consumption, and waiting time, which has better adaptability than manually adjusting the size of the task pool. In the later stage of the solution, when the task pool is full, tasks in the task pool will be assigned to robots using the CMA-ES algorithm to find the optimal task assignment solution for all the robots according to the fitness function including the maximum time and the mean travel distance required by all robots to complete all the tasks. By comparing with fixed threshold method under 56 different task pool sizes, the experimental results show that the handled units can be close to reaching the optimal level, and the average travel distance per unit is lower using adaptive threshold method; so, adaptive threshold solution indeed has better adaptability. This method can satisfy the dynamic and real-time requirements and can be effectively applied to the intelligent warehouse system.

However, because of the complexity and dynamics of the warehouse environment, it may not be accurate to measure the cost by Manhattan distance. Therefore, how to introduce accurate robot motion model to evaluate the cost will be the next work. Furthermore, the relationships among handled units, travel distance per unit, the maximum time taken by all robots to complete all tasks, and the mean distance traveled by all robots need further study. In addition, the effect of communication quality on allocation is not taken into account and will be deeply studied.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

Acknowledgments

This work is supported by the Science and Technology Program of Yantai, China (Grant No. 2019XDHZ085), Major Basic Research Project of Natural Science Foundation of Shandong Province, China (Grant No. ZR2018ZC0438), National Natural Science Foundation of China (Grant No. 61673200), and Laboratory of Robotics in Ludong University.

References

[1] M. Zhou and M. Y. Wang, "Analysis on the development of e-commerce logistics service industry and countermeasures," *Computer and Information Technology*, vol. 20, no. 6, pp. 10–12, 2012.

[2] S. X. Zou, "The present and future of warehouse robot," *Logistics Engineering and Management*, vol. 35, no. 6, pp. 171-172, 2013.

[3] J. J. Enright and P. R. Wurman, "Optimization and coordinated autonomy in mobile fulfillment systems," in *Workshops at the Twenty-Fifth AAAI Conference on Artificial Intelligence*, pp. 33–38, San Francisco, California, 2011.

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

[5] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004.

[6] A. Khamis, A. Hussein, and A. Elmogy, "Multi-robot task allocation: a review of the state-of-the-art," *Eds. Cham: Springer International Publishing*, vol. 604, pp. 31–51, 2015.

[7] B. P. Gerkey and M. J. Matarić, "Sold!: auction methods for multirobot coordination," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 5, pp. 758–768, 2002.

[8] M. Berhault, H. Huang, P. Keskinocak et al., "Robot Exploration with Combinatorial Auctions," *In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 2, pp. 1957–1962, 2003.

[9] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[10] P. MacAlpine, E. Price, and P. Stone, "SCRAM: scalable collision-avoiding role assignment with minimal-makespan for formational positioning," in *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pp. 2096–2102, Austin, Texas, USA, 2015.

[11] L. E. Parker, "ALLIANCE: an architecture for fault tolerant multirobot cooperation," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 2, pp. 220–240, 1998.

[12] B. B. Werger and M. J. Mataric, "Broadcast of local eligibility: behavior-based control for strongly cooperative robot teams," in *Proceedings of the 4th International Conference on Autonomous Agents*, pp. 21-22, Barcelona, Spain, 2000.

[13] Y. Liu, X. Zhang, H. Li, and D. Qian, "Allocating tasks in multi-core processor based parallel system," in *2007 IFIP International Conference on Network and Parallel Computing Workshops*, pp. 748–753, Liaoning, China, 2007.

[14] E. G. Jones, M. B. Dias, and A. Stentz, "Time-extended multi-robot coordination for domains with intra-path constraints," *Autonomous Robots*, vol. 30, no. 1, pp. 41–56, 2011.

[15] J. Zhang and Y. Q. Cao, "Research on dynamic task allocation for MAS based on hybrid genetic and ant colony algorithm," *Computer Science*, vol. 38, no. S1, pp. 268–270, 2011.

[16] J. J. Dou, C. L. Chen, and P. Yang, "Genetic scheduling and reinforcement learning in multirobot systems for intelligent warehouses," *Mathematical Problems in Engineering*, vol. 2015, 10 pages, 2015.

[17] W. Deng, J. Xu, H. Zhao, and Y. Song, "A novel gate resource allocation method using improved PSO-based QEA," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 3, pp. 1737–1745, 2022.

[18] X. Cai, H. Zhao, S. Shang et al., "An improved quantum-inspired cooperative co-evolution algorithm with muli-strategy and its application," *Expert Systems with Applications*, vol. 171, article 114629, 2021.

[19] W. Deng, J. J. Xu, X. Z. Gao, and H. M. Zhao, "An enhanced MSIQDE algorithm with novel multiple strategies for global optimization problems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 52, no. 3, pp. 1578–1587, 2022.

[20] W. Deng, S. Shang, X. Cai et al., "Quantum differential evolution with cooperative coevolution framework and hybrid mutation strategy for large scale optimization," *Knowledge-Based Systems*, vol. 224, article 107080, 2021.

[21] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)," *Evolutionary Computation*, vol. 11, no. 1, pp. 1–18, 2003.

[22] M. Merschformann, L. Xie, and H. Li, "RAWSim-O: a simulation framework for robotic mobile fulfillment systems," *Logistics Research*, vol. 11, no. 8, pp. 1–11, 2018.

[23] L. Xie, N. Thieme, R. Krenzler, and H. Y. Li, *Efficient Order Picking Methods in Robotic Mobile Fulfillment Systems*, 2019, https://arxiv.org/abs/1902.03092.

[24] F. Stulp and O. Sigaud, "Path integral policy improvement with covariance matrix adaptation," in *29th International Conference on Machine Learning*, Edinburgh, Scotland, 2012.

[25] T. Geijtenbeek, M. Van De Panne, and A. F. Van Der Stappen, "Flexible muscle-based locomotion for bipedal creatures," *ACM Transactions on Graphics*, vol. 32, no. 6, pp. 1–11, 2013.

[26] P. MacAlpine and P. Stone, "Overlapping layered learning," *Artificial Intelligence*, vol. 254, pp. 21–43, 2018.

[27] H. R. Zhou, W. S. Tang, and H. L. Wang, "Optimization of multiple traveling salesman problem based on differential evolution algorithm," *Systems Engineering Theory & Practice*, vol. 30, no. 8, pp. 1471–1476, 2010.