Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 597956, 10 pages
http://dx.doi.org/10.1155/2015/597956

Research Article

Genetic Scheduling and Reinforcement Learning in Multirobot Systems for Intelligent Warehouses

Jiajia Dou,[1,2] Chunlin Chen,[1,2] and Pei Yang[1,2]

[1]*Department of Control and Systems Engineering, School of Management and Engineering, Nanjing University, Nanjing 210093, China*
[2]*Research Center for Novel Technology of Intelligent Equipments, Nanjing University, Nanjing 210093, China*

Correspondence should be addressed to Chunlin Chen; clchen@nju.edu.cn and Pei Yang; yangpei@nju.edu.cn

Received 8 August 2015; Accepted 15 December 2015

Academic Editor: Luciano Mescia

A new hybrid solution is presented to improve the efficiency of intelligent warehouses with multirobot systems, where the genetic algorithm (GA) based task scheduling is combined with reinforcement learning (RL) based path planning for mobile robots. Reinforcement learning is an effective approach to search for a collision-free path in unknown dynamic environments. Genetic algorithm is a simple but splendid evolutionary search method that provides very good solutions for task allocation. In order to achieve higher efficiency of the intelligent warehouse system, we design a new solution by combining these two techniques and provide an effective and alternative way compared with other state-of-the-art methods. Simulation results demonstrate the effectiveness of the proposed approach regarding the optimization of travel time and overall efficiency of the intelligent warehouse system.

1. Introduction

Intelligent warehouses are an essential part of the material-handling industry and have gained popularity in recent years [1]. Efficient and sensitive warehousing with intelligent mobile robots is critical to improve the overall productivity and simultaneously achieve high efficiency. The operations with respect to warehousing systems have received considerable and increasing attention [2, 3]. In this paper, we focus on the path planning problem of autonomous mobile robots and the scheduling issues of the task orders. The path planning problem [4] is to generate a collision-free path from the starting point to the ending point and it is recognized as a fundamental issue in warehousing systems. Task scheduling [5] is the method to assign tasks to the robots under certain criteria.

The scheduling and path planning problems are a hot topic in multirobot systems for warehouses and many effective methods have been developed over the past few decades. In 1996, Jennings [6] introduced a task negotiation approach for groups of autonomous robots in warehousing systems. However, the coordination approach presented in that paper cannot be applied to the uncertain environment. The combinatorial auction algorithm proposed by Sandholm [7] is considered to be more efficient than traditional mechanisms as it divides tasks into different clusters based on the correlations between them. Then the robots bid on the unrestricted combinations of tasks. A path planning approach combining the fuzzy method with the adaptive Q-leaning algorithm was proposed in [8] and it was merely adapted to robot soccer systems. In 2004, Richards et al. [9] used a standard A^* search algorithm to find the optimal path. Nevertheless, the method is only for the controlled and known environment.

Most previous research of warehousing operations concentrated on minimizing the travel distance where the workload among the robots is not well balanced and the utilization of multirobot systems is not fully optimized. Few publications could be found about the travel time. In 2011, Elango et al. [10] used the K-means clustering and auction-based mechanisms to assign tasks to the robots in a balanced manner. This approach takes two objectives into consideration, that is, total travel cost minimization and the workload balancing among the robots. However, the complexity of the algorithm is comparatively high. Zhou et al. [11] proposed a balanced

heuristic mechanism to solve the task allocation problem in an intelligent warehouse, which improves the balance performance among the robots and reduces the total travel time effectively.

Unfortunately, even if there have been considerable researches on scheduling and path planning problems, combinations of these two aspects have rarely been studied in detail. In many cases, these two issues are even considered to be independent. The truth is that the assignment results from the path-length-based scheduling process and in turn affects the performance of the path planning approach. However, this relationship is often ignored in the previous researches. Our work considers the two issues simultaneously. We employ reinforcement learning (RL) to explore the dynamic and unknown circumstance and use genetic algorithm (GA) to generate balanced and efficient task assignments for the robots. We provide three criteria [11] to evaluate the proposed algorithms, that is, travel time (TT), total travel cost (TTC), and balancing utilization (BU). We demonstrate the effectiveness of the proposed methods regarding the optimization of travel time and overall efficiency for the warehousing system through simulations.

The rest of this paper is organized as follows. In Section 2, an introduction to the intelligent warehousing systems is given, including the basic structures and the mathematic models. In Section 3, we discuss the scheduling and path planning problems in warehousing systems. Then the proposed methods combining GA and RL are described in detail. In Section 4, we elaborate on the implementations of the new solution in an intelligent warehouse system as well as comparisons with some other relevant algorithms. Then the experimental results are presented and analyzed. Finally, we conclude this paper in Section 5.

2. Problem Statement

In this paper, we focus on the task allocation and path planning problems for the intelligent warehouses with multirobot systems. In a practical warehouse (e.g., the Kiva Mobile Fulfillment System [12, 13]), there are generally dozens or even hundreds of mobile robots grabbing and moving shelves directly to workers at the picking station according to the actual task orders. In order to simplify the system, we make the following assumptions about the robots, tasks, and the environment:

(i) A grid-based map model is used for the warehouse environment.

(ii) The environment of the system is unknown to the robots.

(iii) The robot is equipped with necessary sensors that can detect the environment.

(iv) The robot can get to an adjacent site through one of the four actions: up, down, left, and right.

(v) Both the robots and tasks are identical with each other.

(vi) The robots are essentially independent when performing the tasks.

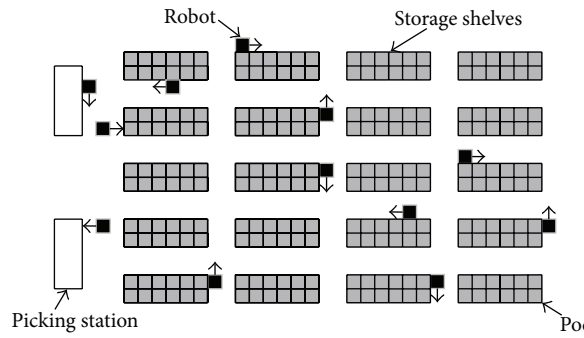

Figure 1: Simplified configuration of an intelligent warehouse system.

(vii) The total cost and the whole time to complete the task are measured by the walking steps of the robots.

(viii) The elapsed time at the station and lifting the shelf is fixed and constant.

The simplified model of a specific intelligent warehouse is shown in Figure 1. The system is composed of storage shelves, robots, and picking stations. Each storage shelf contains several pods and each of them holds a certain number of products. The robot lifts and carries a corresponding pod to the picking station along a learned path and then returns the pod to the original position based on the allocated orders. It is clear that both task allocation and path planning algorithms are crucial in getting the work done as soon as possible.

Task allocation is the method to assign the unfulfilled task orders to the robots. It is one of the most critical problems in warehouse systems. Path planning for a single robot can be defined as searching for an optimal or suboptimal path from the starting point to the destination regarding distance and time, which is considered to be a fundamental issue in warehousing systems. Furthermore, these two problems are closely correlated with each other and their relationship is taken into account in this paper. We emphasize these two issues simultaneously and attempt to accomplish the tasks as quickly as possible to improve the efficiency of warehousing systems.

In a typical warehousing system, there always exist numerous unfulfilled task orders. First we are confronted with a problem of how to assign the available tasks to a limited number of working robots. By making wise decisions about task scheduling we can reduce the time spent to get the work done. Afterwards, when the robots start to execute their individual tasks we should consider the path planning problem. The goal is to find a collision-free path to the destination and avoid obstacles with as few steps as possible.

It is clear that at a high level our ultimate goal is to keep the robots as busy as possible and maximize the efficiency of the whole system. Thus several relevant targets need to be taken into account. The first objective we considered is reducing the amount of time spent. As the simulated running time is closely related to the performance of computer systems, the running time is measured by the largest walking steps among the robots. The second objective is to minimize

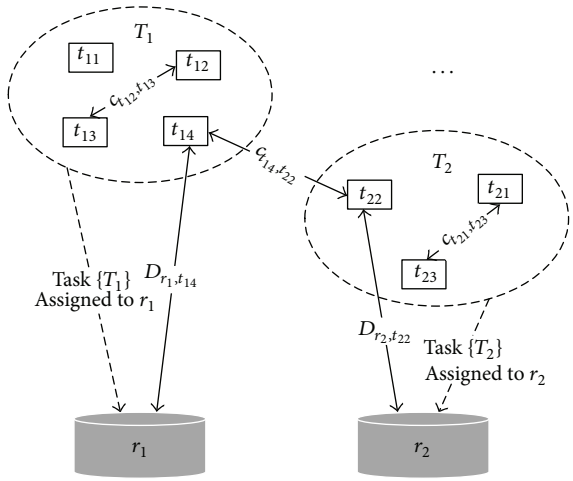

FIGURE 2: Basic parameters and indexes for an intelligent warehouse with multiple robots.

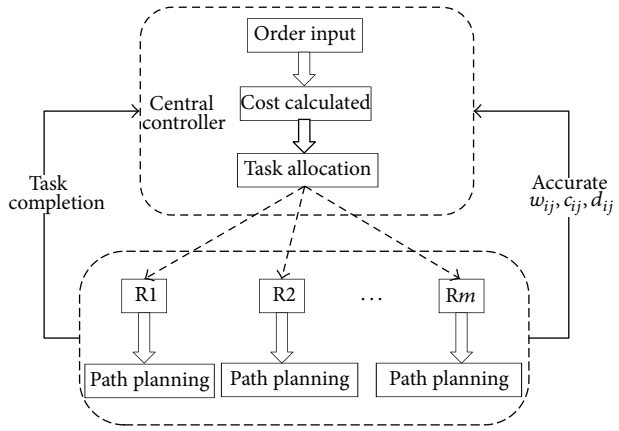

FIGURE 3: Overall control structure.

the total distances of all the actual routes. In addition, in order to balance the workload of the autonomous robots, the balancing condition is taken into consideration as well.

We assume that there are a set of n tasks $T = \{t_1, t_2, \ldots, t_n\}$ to be performed by a set of m robots $R = \{r_1, r_2, \ldots, r_m\}$. The travel cost between each pair of distinct tasks $\{t_i, t_j\}$ $(t_i, t_j \in T)$ is represented as c_{ij}. Apparently, the cost satisfies $c_{ij} = c_{ji}$ in all cases. The travel cost between a robot r_i and a task t_j is expressed as d_{ij}. We define w_i as the cost of task t_i, which is the total cost of carrying the inventory pod to the picking station according to task t_i and then taking it back. The values of c_{ij}, d_{ij}, and w_i are all calculated by path planning algorithms which depend only on the actual path lengths.

Suppose the tasks are divided into m groups $T = \{T_1, T_2, \ldots, T_m\}$ and each section T_i is allocated to robot r_i. We assume that r_i have k tasks to perform $T_{i1} = \{t_{i1}, t_{i2}, \ldots, t_{ik}\}$; then the total cost of all the tasks in T_i can be expressed as $W(r_i) = w_{i1} + w_{i2} + \cdots + w_{ik}$. The basic parameters and indexes used in the model are introduced in detail as shown in Figure 2.

We define the individual travel cost $\mathrm{ITC}\{r_i, T_i\}$, which indicates the total travel cost for robot r_i to complete its assigned task set T_i and it can be computed by

$$\mathrm{ITC}(r_i, T_i) = D_{r_i t_{i1}} + \sum_{j=1}^{k-1} c_{t_{ij} t_{i(j+1)}} + W(r_i), \tag{1}$$

where $D_{r_i t_{i1}}$ represents the travel cost for robot r_i to reach its first assigned task t_{i1} and $\sum_{j=1}^{k-1} c_{t_{ij} t_{i(j+1)}}$ is the total travel cost for the other $(k-1)$ tasks.

In this paper, three criteria are used to evaluate the performance of different solutions: travel time (TT), which represents the maximum of all the individual travel costs among the robots; total travel cost (TTC), which indicates the total travel cost of all the robots in the system; and balancing utilization (BU), which assesses the balancing condition of the whole system. The corresponding formulas [11] are listed as follows:

$$\begin{aligned} \mathrm{TT} &= \max_i \mathrm{ITC}(r_i, T_i), \\ \mathrm{TTC} &= \sum_{i=1}^{m} \mathrm{ITC}(r_i, T_i), \\ \mathrm{BU} &= \frac{\min_i \mathrm{ITC}(r_i, T_i)}{\max_i \mathrm{ITC}(r_i, T_i)}. \end{aligned} \tag{2}$$

3. GA and RL for Warehouse Scheduling and Multirobot Path Planning

3.1. Overall Control Structure. In this paper, the workspace is divided into numerous grid cells with storage shelves in the middle and picking stations around the perimeter. The robots are working on this rasterized map. When an order is received, the robot retrieves the appropriate pod and brings it to the worker standing at inventory stations. Then the appropriate items are picked, packed, and shipped out of the warehousing system.

Throughout the above process, the task allocation process is executed by the central controller, whereas the path planning is implemented by the robots. During the task allocation process, the central controller receives a set number of task orders from the Task Pool and employs GA [14–17] to assign tasks to the robots. After that, the central controller transmits the assignment results to the robots via wireless communication and the robots execute the tasks progressively by using Q-learning [18–23] (a widely used RL algorithm) to plan paths all along since the environment is unknown. Figure 3 gives a brief overview of the whole process.

Another point that must be noted is the collision avoidance between robots since the reinforcement learning algorithm in this paper is only for a single robot. The interaction which happens when the path of one robot is blocked by another robot or two or more robots attempt to move into the same cell at a given time is not only variable but also unpredictable. A straightforward mechanism that may tackle this problem is to refrain from these collisions. When robots

```
(1)  Initialize parameters: population size Popsize, maximal generations MaxEpoc, crossover rate Pc;
(2)  Calculate the distances between different task orders;
(3)  Generate an population of feasible solutions randomly;
(4)  Evaluate individuals in the initial population;
(5)  for i = 1 to MaxEpoc do
(6)      Set the alterable mutation rate Pm according to the actual evolution generations;
(7)      Select individuals according to elitist strategy and roulette wheel method;
(8)    if random (0, 1) < Pc then
(9)        Crossover individuals in pairs by a variation of the order crossover operator;
(10)   end if
(11)   if random (0, 1) < Pm then
(12)       Mutate an individual by swap mutation;
(13)   end if
(14)   Evaluate the produced offspring;
(15) end for
```

Algorithm 1: Genetic algorithm for warehouse scheduling.

detect the others at short range (usually a few grids in front of them), they enter into negotiation and, in general case, the robot with fewer tasks left gives up the right-of-way. This mechanism is critical to the security of the whole system.

3.2. GA Based on Virtual Task for Warehouse Scheduling. GA is a search method inspired from natural evolution and has been successfully applied to a variety of NP-hard combinatorial optimization problems [14–17]. It is iterative, population-based algorithm and follows the principle of survival of the fittest to search solutions through the operations of selection, crossover, and mutation. To apply GA to solve the warehouse scheduling problem, the focus is to establish an effective chromosome representation and design suitable selection, crossover, and mutation operators to improve the solution quality.

In this paper, a concept of virtual task is introduced to code the multirobot task allocation problem; meanwhile evaluation function and genetic operators are also specifically designed for the issue. An improved GA for warehouse scheduling is presented in Algorithm 1. The entire algorithm terminates when a fixed evolution generation is reached.

3.2.1. Chromosome Representation. Chromosome representation is particularly important in GA and a specific problem must be indicated by the appropriate code to promote the evolutionary process. In this paper, we use a single integer string of length $(m+n-1)$ as representation of chromosome, where m is the number of robots and n is the number of tasks. In the chromosome, n tasks are represented by a permutation of integers from 1 to n. The string is portioned into m sections by adding $(m-1)$ virtual tasks, that is, $(n+1), (n+2)\cdots(n+m-1)$, which indicate the transition from one robot to the other.

An example of task allocation for 8 tasks with 3 robots is given to demonstrate the above chromosome representation measure. We only need to add 2 virtual tasks in the chromosome, that is, 9 and 10, to be the division points. The sequences in Table 1 are the possible situations for the task allocation.

During the process of evolution, the virtual tasks in the chromosome may appear at both ends or be adjacent to each other. Hence there will be at least one robot who has no tasks to perform, which will lead to unbalance in the warehouse; thus this kind of chromosome will be eliminated during the evolution.

Table 1: Task allocation coding and the corresponding distribution results.

Chromosomes	Distribution results		
3 5 6 9 2 7 10 1 4 8	R1: 3 5 6	R2: 2 7	R3: 1 4 8
9 3 4 8 10 2 1 6 5 7	R1: 0	R2: 3 4 8	R3: 2 1 6 5 7
2 5 7 10 9 1 8 4 6 3	R1: 2 5 7	R2: 0	R3: 1 8 4 6 3
6 3 1 4 9 2 5 8 7 10	R1: 6 3 1 4	R2: 2 5 8 7	R3: 0

3.2.2. Design of Fitness Function. Fitness functions are used in GA to evaluate the quality of the candidate solutions. When the initial population is generated, or an offspring is produced, the fitness values of the feasible solutions are calculated by a fitness function. The higher the fitness value is, the better the individual is. In other words, a fitness function can facilitate the evolution event by distinguishing the good solutions from the bad ones. However, there is no such benchmark for a fixed fitness function. We need to design an appropriate fitness function based on the characteristics of our warehousing systems.

In the case of warehouse scheduling, we aim to achieve three different objectives: (1) minimizing the overall time to perform the tasks, TT, (2) minimizing the total travel cost of all the robots, TTC, (3) and maximizing the balancing condition of task allocation, BU. Hence, the fitness function includes three different parts accordingly. For the first two parts, the reciprocals of TT and TTC are used to be the evaluation criteria; thus the minimum solutions are converted to the maximum ones. Taking MaxTest and TotalTest as the two constants related to the actual task numbers, the fitness function can be defined as

$$F(x) = \frac{\text{MaxTest}}{\text{TT}} + \frac{\text{TotalTest}}{\text{TTC}} + \text{BU}. \tag{3}$$

3.2.3. The Genetic Operators

(a) Selection Operator. Selection selects a certain number of individuals based on the fitness value. In general, the whole process follows the principle of survival of the fittest. In other words, the individuals with higher fitness values are more likely to be selected to the next generation than the others. In this paper, we combine elitist strategy with roulette wheel method as the selection strategy. First, we sort the individuals at present according to their own fitness value and copy the top 20% directly to the next generation. This elitist strategy can prevent the outstanding individuals from being eliminated after a selection operator which is a vital guarantee for the convergence of the algorithm. Then roulette wheel method is used to choose the remaining individuals. Roulette wheel method is also named as proportional model and the possibility of being selected is proportional to the fitness of each individual. The better the individuals are, the more chances to be inherited to the next generation they have. This new selection strategy is proved to be valid and efficient during the evaluation.

(b) Crossover Operator. After selection, individuals are crossed over to create various and promising offspring. It is commonly recognized that crossover operation is the key method to generate new individuals. Furthermore, it is an important feature which distinguishes GA from other evolutionary algorithms. Similarly, the top 20% individuals are directly inherited to the next generation. For the rest, a variation of the order crossover (OX) [14] operator is introduced.

Our crossover operator is responsible for the recombination of two individuals to form two new offspring. Thus, the new individuals will inherit some of the features from the first parent and retain some characteristics of the other. Specifically, the new individuals are constructed in the following way. Regardless of those outstanding individuals who will not be crossed over, the remainders are paired as the parent individuals in accordance with the sequences in the population. After that, for each pair, two crossing points need to be determined. Unlike the traditional methods, we select the first crossing point in the anterior half of the individual and the second in the latter half. This measure has well controlled the length of the subsection determined by the two cut points which will influence the evolution rate and the convergence of the algorithm. Finally, we construct new individuals by coping with the subsection of one parent and retaining the relative order of genes of the other one.

For example, if there are two parent individuals,

$$\begin{aligned} &(9\ \ 3\ \ 4\ \ 8\ \ 10\ \ 2\ \ 1\ \ 6\ \ 5\ \ 7),\\ &(2\ \ 5\ \ 7\ \ 10\ \ 9\ \ 1\ \ 8\ \ 4\ \ 6\ \ 3), \end{aligned} \tag{4}$$

suppose that the substrings between the vertical lines are selected:

$$\begin{aligned} &(9\ \ 3\ \ 4\ \mid\ 8\ \ 10\ \ 2\ \ 1\ \mid\ 6\ \ 5\ \ 7),\\ &(2\ \ 5\ \ 7\ \mid\ 10\ \ 9\ \ 1\ \ 8\ \mid\ 4\ \ 6\ \ 3). \end{aligned} \tag{5}$$

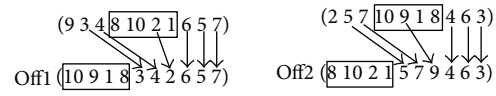

Figure 4: The crossover procedure.

The corresponding segments are directly duplicated into the offspring, respectively, which result in

$$\begin{aligned} &(8\ \ 10\ \ 2\ \ 1\ \ *\ \ *\ \ *\ \ *\ \ *\ \ *),\\ &(10\ \ 9\ \ 1\ \ 8\ \ *\ \ *\ \ *\ \ *\ \ *\ \ *). \end{aligned} \tag{6}$$

Next, copy the genes that are not presented in the offspring yet from the other parent in the order they appear in. In this example, two new offspring are produced:

$$\begin{aligned} &(8\ \ 10\ \ 2\ \ 1\ \ 5\ \ 7\ \ 9\ \ 4\ \ 6\ \ 3),\\ &(10\ \ 9\ \ 1\ \ 8\ \ 3\ \ 4\ \ 2\ \ 6\ \ 5\ \ 7). \end{aligned} \tag{7}$$

The procedure of the above operators is described in Figure 4.

(c) Mutation Operator. Mutation takes charge of adding new traits to the individuals, thus maintaining variability of the population. It is indispensable in preventing premature termination in GA by improving the local search ability. It should be noted that, in order to preserve the current excellent individuals, the top 20% do not participate in the mutation operation. We adopt swap mutation (SM) [14] operator in this task scheduling algorithm for the other individuals. The swap mutation operator which is also named as the point mutation operator randomly selects two genes in the chromosome and exchanges them. For example, consider the following sequence:

$$(5\ \ 3\ \ 6\ \ 9\ \ 2\ \ 7\ \ 10\ \ 1\ \ 4\ \ 8), \tag{8}$$

and assume that the second and the fifth genes are randomly selected; we swap the two points simultaneously and find that it turns into

$$(5\ \ 2\ \ 6\ \ 9\ \ 3\ \ 7\ \ 10\ \ 1\ \ 4\ \ 8). \tag{9}$$

In this operation we swap the characteristics of individuals governed by the mutation rate at a certain extent. An appropriate mutation rate may ensure the ability to explore the entire search space without weakening the performance of the algorithm. Thus, we select different values according to different requirements. Initially, the mutation operation is performed with a low probability of 0.1 to preserve the vital individuals. After 1000 generations, the evolution process tends to be stable, and we reset the mutation rate at a higher value of 0.5 to search for the new individuals.

3.3. Q-Learning for Path Planning of Multirobot Systems. Reinforcement learning (RL) [23–26] is a widely used learning method in cybernetics, statistics, mental philosophy, and so forth. It has been an active branch in AI for decades. RL is a powerful mechanism that does not require any prior

(1) Initialize the total episodes of learning Maxepi, maximum step number Maxstep, discount rate γ;
(2) Initialize the coordinate of the starting and ending point;
(3) Initialize $Q(s, a)$ based on the actual grid world;
(4) **for** $i = 1$ to Maxepi **do**
(5) Initialize the learning rate α;
(6) **for** $j = 1$ to Maxstep **do**
(7) Decrease the learning rate α gradually;
(8) Choose a from s by greedy action selection;
(9) Take action a_t, observe r_{t+1}, s_{t+1};
(10) $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_{a \in A} Q(s_{t+1}, a) - Q(s_t, a_t)]$
(11) $s \leftarrow s_{t+1}$
(12) **if** s_{t+1} is a goal state **then**
(13) Break;
(14) **end if**
(15) **end for**
(16) **end for**

Algorithm 2: Q-learning for path planning.

knowledge and the robot can autonomously learn to improve its performance over time by the trial-and-error interacting with its working environment. The only response from the environment is a delayed reward, and the goal of the robot is to maximize the accumulated rewards in the long run for each action. In this kind of learning scheme, the system designers just need to provide the ultimate goal to be achieved instead of telling the robot how to reach the target. On account of the above characteristics, RL has become a hotspot of research and is a promising method to solve the path planning problem for mobile robots.

The particular RL algorithm that we employ for the problem of path planning here is Q-learning. Q-learning [18–23] is a kind of model-free RL algorithms which has been widely used due to its simplicity and theory maturity. Q-learning uses the action-value function $Q(s, a)$ to indicate the estimated values for each state-action pair and then recursively finds the optimal solution. The procedural form of Q-leaning for path planning is presented in Algorithm 2. Several major issues about Algorithm 2 are addressed as follows.

3.3.1. The Reward Function. The robots proceed their learning course through the rewards they obtained after a certain action is taken. The reward value is a feedback from the environment which evaluates the action in an immediate sense. A reward function indicates how good the robot's performance is and thus defines the goal in RL problem. In this paper, a frequently employed reward strategy is adopted. In the grid world, the purpose of the robot is to find an optimal collision-free path from the starting point to the ending point. Therefore, upon reaching the corresponding goal, the robot will attract a large positive reward of 200. A negative reward of −50 is received as a punishment when the robot collides into a wall. In other cases, a reward of −1 is provided so as to encourage fewer steps. In this way, the robot will prefer those actions which produce higher rewards over the long run in the process of learning. In other words, this reward function will prompt the robot to follow the optimal path that results in the best possible accumulated rewards.

3.3.2. The Value Function. Unlike the reward function, the value function picks out those actions which obtain for us the maximal rewards over the long run. The target of the learning robot is to maximize the accumulated rewards per action it receives from the environment. Q-learning finds an optimal strategy by learning the values of the so-called Q function. The function $Q(s, a)$ is defined as the expected discounted cumulative reward that is received by the robot when executing action a in state s and performing an optimal strategy afterwards.

We start with an initial estimation of $Q(s, a)$ for each state-action pair based on the actual grid world and the value is updated by interacting with the dynamic environment continuously. In the course of approaching the goal, the next state of the environment is determined by the current state and the action the robot takes. At a certain step t, the robot acquires the current state of the environment $s_t \in S$, where S is the set of all possible states, and then takes an action $a_t \in A$, where A is the set of actions available in state s_t. After the action a is taken, an immediate reward r_{t+1} of the state-action pair is sent to the robot and the environment is in a new state s_{t+1}. The corresponding updated rule [23] is as follows:

$$\begin{aligned} &Q(s_t, a_t) \\ &\quad \longleftarrow Q(s_t, a_t) \\ &\qquad + \alpha \left[r_{t+1} + \gamma \max_{a \in A} Q(s_{t+1}, a) - Q(s_t, a_t) \right], \end{aligned} \tag{10}$$

where $\max_{a \in A} Q(s_{t+1}, a)$ is the maximum Q-value calculated for taking all possible actions on the next state s_{t+1}. The parameter $\gamma \in [0, 1]$ represents the discount factor that controls how much impact future rewards have on the present decision-making. With a small value of γ, the robot tends to attach more importance to the near-term rewards, whereas the robot will take the future rewards into consideration as γ becomes closer to 1. The parameter $\alpha \in [0, 1]$ is the learning rate. A small value of this parameter means the learning process will proceed at a slow speed, while a too high value

might affect the convergence property of the algorithm. Obviously, an appropriate learning rate is critical to the performance of the whole algorithm. In this paper, the learning rate is alterable and decreased step by step during the process of learning. Finally, it will approach a relative small value. The action-value function $Q(s, a)$ is updated at each time step of an episode until the robot enters into the goal state. It has been proved that if all state-action pairs are visited infinitely often by the robot and an appropriate learning rate is selected, the estimated $Q(s, a)$ will converge with probability one to the optimum value $Q^*(s, a)$.

3.3.3. Action Selection Strategies. In the physical world, a robot usually can move in all directions. Here, the action space is simplified by assuming that only five basic movements are available for each robot: up, down, left, right, and stand. At each time step, the robot can select any of them as long as there is an admissible transition. When a robot tries to implement an action that would take it out of bounds, the motion will not happen.

In the path planning problem, in order to find a collision-free path within the least amount of steps, the robot needs to employ as many optimal action selection strategies as possible in the process of learning. The task of a robot is thus to learn a policy π that maps the current state s into the desirable action a to be performed. The action policy should be learned through trial-and-error and through evaluating every state-action pair so that we can build a policy for working in the intelligent warehouse. During the process of learning, if the action is selected according to the random principle, some unnecessary pairs which have little Q-value will occur frequently. This can bring long learning time and the Q-learning algorithm may not converge. In this paper, we use a deterministic selection mechanism called greedy policy to choose actions. A greedy strategy [23] refers to a policy that the robot invariably sorts out the action with the highest estimated Q-value:

$$\pi^*(s) = \arg\max_a Q(s, a). \tag{11}$$

It assigns probability one to action $\arg\max_a Q(s, a)$ in state s and the selected action is called a greedy action. The experiments show that the action selection strategy is quite effective and the learning speed is increased evidently.

4. Experiments

In this section, we test the performance of the proposed integrated approach by applying it to a simulated intelligent warehouse system. As shown in Figure 5, the map of the warehouse is divided into a 50×28 grid world where each robot and pod occupies a single grid exactly. There are 4×12 shelves in total located in the center and each storage shelf consists of 6 inventory pods. Three picking stations are situated in the bottom side of the whole system. We have 8 robots available working for the task orders and each time they can move just one cell within the scope of the map.

A Task Pool is used to store the dynamic entry-orders and it will send a certain number of orders to the central controller at a specific time. After the controller receives the tasks from the Task Pool, we apply various methods to assign them to the robots. Then, the central controller sends the assignment results to the robots via wireless communication and the robots start to work on their own tasks by using path planning algorithms to calculate the path all the way.

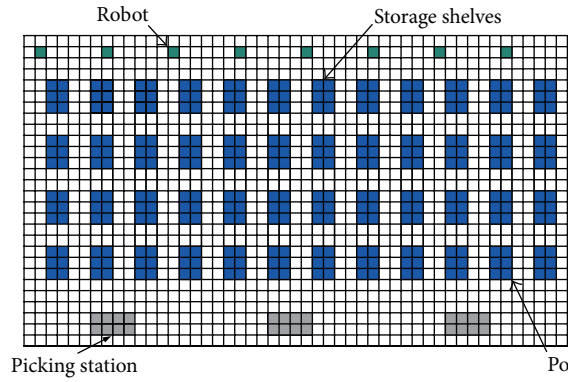

Figure 5: Rasterized map of the simulated intelligent warehouse system.

4.1. Parameter Settings. The parameter settings for GA are selected as follows: the population size is set to 100; mutation probability is primarily assigned to 0.1 and then transformed into 0.5 after 1000 generations; the crossover rate is set to 0.95, which is usually kept high to increase the global search ability; the total number of generations is assigned to be a large integer, 100000, to get a satisfactory result. Parameter settings for Q-learning are very critical since they have a direct impact on the rate of convergence and quality of the final results. The discount factor is set to a constant number, 0.8, while the learning rate is typically time varying. The initial value of learning rate is assigned to 0.3 and it decays by each step during the leaning process. Finally, the learning rate will reach a small value at the end of run. The maximum step number for each episode is set as 15000. When the number of the accumulated steps reaches 15000, it means failure to reach the target within the limited number of steps and the episode will be terminated and continued to the next one. The maximum episodes for Q-learning are set as 8000 to obtain reliable experimental results.

4.2. Experimental Results. In order to verify the effectiveness of the proposed approach, we carried out several groups of experiments by combining different task allocation methods (i.e., GA, auction-based method [7, 11], K-means clustering method [10], and random allocation) with different path planning methods (i.e., Q-learning and classic heuristic algorithm of A^* [9, 11]), which leads to eight groups of simulation experiments in the warehousing system as follows:

(1) GA with Q-learning.

(2) GA with A^*.

(3) Auction with Q-learning.

(4) Auction with A^*.

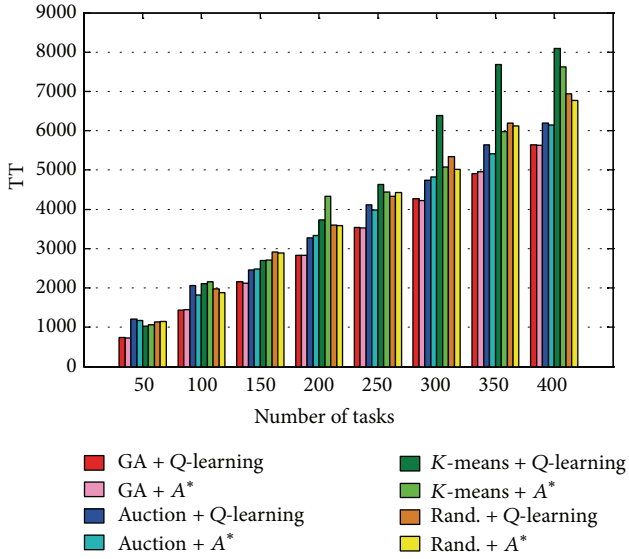

Figure 6: Travel time (TT) spent by each different approach.

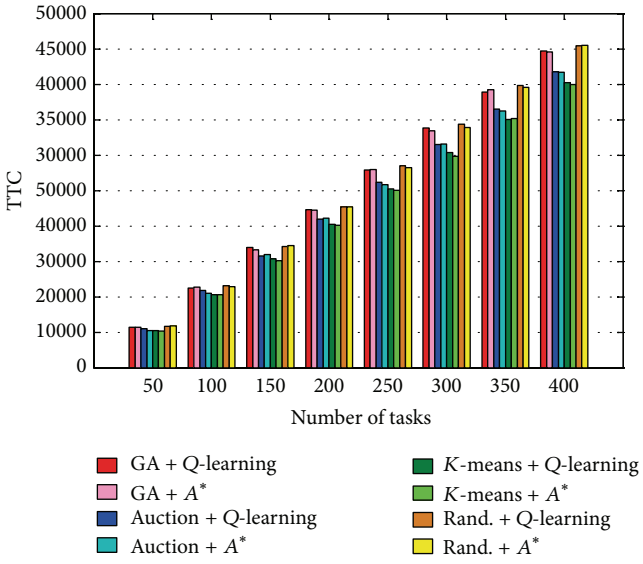

Figure 7: Total travel cost (TTC) of each different approach.

(5) K-means with Q-learning.

(6) K-means with A^*.

(7) Rand. with Q-learning.

(8) Rand. with A^*.

The results of all these solutions are shown in Figures 6–8, where the performances are given regarding the criteria of travel time, TT, total travel cost, TTC, and balancing utilization, BU. All the results are averaged over 20 simulations in the above experimental environment.

Figure 6 shows the time spent by the above eight groups of experiments as the number of tasks increases. It can be seen that GA performs much better than the others where the robots are able to accomplish the allocation task in a much shorter time, while the performance of K-means is the worst. This is because, in the traditional K-means method, the task allocation process concentrates more on minimizing the travel cost and cannot control the balancing condition effectively. Once there is a robot nearest to an open task, other robots will not attempt to compete even if they are sitting around idle and thus caused a bad behavior.

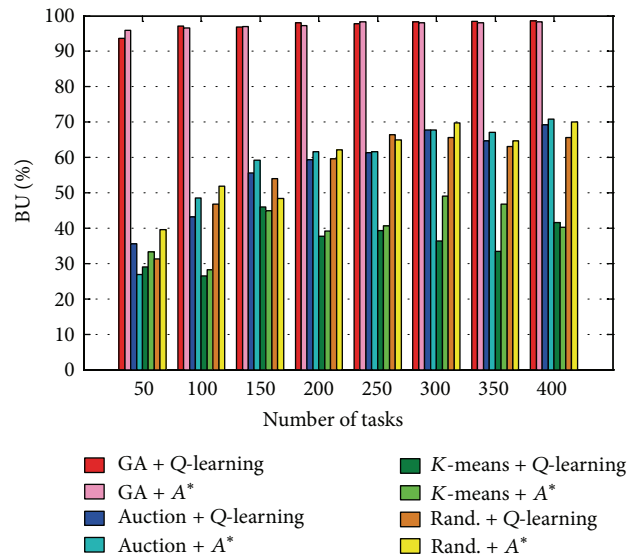

Figure 8: Balancing utilization (BU) of each different approach.

Figure 7 displays the curves for the total travel cost. It is clear that there is only a slight distinction between these eight groups regardless of the number of tasks. Unsurprisingly, the travel costs of K-means are the lowest among all the others. Another important aspect is that Q-learning exhibits a similar performance with A^*. In other words, although Q-learning does not use the model of the environment, it can find the optimal path as A^* does.

Figure 8 shows the balancing utilization and the performance of GA combined with A^* or Q-leaning are strikingly different from the other solutions. They have a steady value of about 95% all along. By contrast, K-means combined with A^* or Q-leaning perform significantly worse. The BU of them is much smaller; none of them reached 50% which indicates a severe bias towards using of robots and poor balancing performance. In that condition, the utilization of the system is quite unsatisfied. The other four groups outperform K-means in most cases, the BU values range from 25% to 75% as the task number increases, and the four methods perform very close to each other.

5. Conclusion

In this paper, we focus on the typical scheduling and path planning problems in an intelligent warehouse. GA and Q-learning are presented for task allocation and path planning, respectively. In comparison, we review and evaluate some popular algorithms in these two issues for autonomous mobile robots in warehousing systems. The experimental results demonstrate the effectiveness of the new solution by combining GA and RL. GA generates the best results

for warehouse scheduling compared to other methods we examined. The performance of A^* is roughly the same with Q-learning for path planning problem. However, A^* is only suitable for the case when the working environment is known to the robots. As RL does not necessarily need the model of the environment, the robot gains the ability to adapt to different circumstance. The new solution improves the utility of the intelligent warehouse system remarkably and is practical for various applications. Our further work will focus on avoiding dynamic obstacles in a warehouse and transfer learning for multiple robots [27–29].

Conflict of Interests

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

Acknowledgment

This work was supported by the National Natural Science Foundation of China (nos. 61273327 and 61432008).

References

[1] R. de Koster, T. Le-Duc, and K. J. Roodbergen, "Design and control of warehouse order picking: a literature review," *European Journal of Operational Research*, vol. 182, no. 2, pp. 481–501, 2007.

[2] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI Magazine*, vol. 29, no. 1, pp. 9–19, 2008.

[3] J. Gu, M. Goetschalckx, and L. F. McGinnis, "Research on warehouse operation: a comprehensive review," *European Journal of Operational Research*, vol. 177, no. 1, pp. 1–21, 2007.

[4] T.-K. Wang, Q. Dang, and P.-Y. Pan, "Path planning approach in unknown environment," *International Journal of Automation and Computing*, vol. 7, no. 3, pp. 310–316, 2010.

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

[6] N. R. Jennings, *Coordination Techniques for Distributed Artificial Intelligence*, Wiley-Interscience, 1996.

[7] T. Sandholm, "Algorithm for optimal winner determination in combinatorial auctions," *Artificial Intelligence*, vol. 135, no. 1-2, pp. 1–54, 2002.

[8] K.-S. Hwang, S.-W. Tan, and C.-C. Chen, "Cooperative strategy based on adaptive Q-learning for robot soccer systems," *IEEE Transactions on Fuzzy Systems*, vol. 12, no. 4, pp. 569–576, 2004.

[9] N. D. Richards, M. Sharma, and D. G. Ward, "A hybrid A*/automaton approach to on-line path planning with obstacle avoidance," in *Proceedings of the 1st AIAA Intelligent Systems Technical Conference*, pp. 141–157, Chicago, Ill, USA, September 2004.

[10] M. Elango, S. Nachiappan, and M. K. Tiwari, "Balancing task allocation in multi-robot systems using K-means clustering and auction based mechanisms," *Expert Systems with Applications*, vol. 38, no. 6, pp. 6486–6491, 2011.

[11] L. Zhou, Y. Shi, J. Wang, and P. Yang, "A balanced heuristic mechanism for multirobot task allocation of intelligent warehouses," *Mathematical Problems in Engineering*, vol. 2014, Article ID 380480, 10 pages, 2014.

[12] R. D'Andrea and P. Wurman, "Future challenges of coordinating hundreds of autonomous vehicles in distribution facilities," in *Proceedings of the IEEE International Conference on Technologies for Practical Robot Applications (TePRA '08)*, pp. 80–83, Woburn, Mass, USA, November 2008.

[13] J. Enright and R. Wurman, "Optimization and coordinated autonomy in mobile fulfillment systems," in *Proceedings of the Workshops at the 25th AAAI Conference on Artificial Intelligence*, pp. 33–38, San Francisco, Calif, USA, August 2011.

[14] P. Larrañaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, and S. Dizdarevic, "Genetic algorithms for the travelling salesman problem: a review of representations and operators," *Artificial Intelligence Review*, vol. 13, no. 2, pp. 129–170, 1999.

[15] M. F. Tasgetiren, P. N. Suganthan, Q.-K. Pan, and Y.-C. Liang, "A genetic algorithm for the generalized traveling salesman problem," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '07)*, pp. 2382–2389, Singapore, September 2007.

[16] J. Li, Q. Sun, M. C. Zhou, and X. Dai, "A new multiple traveling salesman problem and its genetic algorithm-based solution," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC '13)*, pp. 627–632, IEEE, Manchester, UK, October 2013.

[17] A. Király and J. Abonyi, "Optimization of multiple traveling salesmen problem by a novel representation based genetic algorithm," in *Intelligent Computational Optimization in Engineering*, vol. 366 of *Studies in Computational Intelligence*, pp. 241–269, Springer, Berlin, Germany, 2011.

[18] C. Chen, D. Dong, H. X. Li, and T. J. Tarn, "Hybrid MDP based integrated hierarchical q-learning," *Science in China Series F—Information Sciences*, vol. 54, no. 11, pp. 2279–2294, 2011.

[19] A. Konar, I. G. Chakraborty, S. J. Singh, L. C. Jain, and A. K. Nagar, "A deterministic improved Q-learning for path planning of a mobile robot," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 43, no. 5, pp. 1141–1153, 2013.

[20] C. Chen, H.-X. Li, and D. Dong, "Hybrid control for robot navigation—a hierarchical Q-learning algorithm," *IEEE Robotics and Automation Magazine*, vol. 15, no. 2, pp. 37–47, 2008.

[21] C. Chen, D. Dong, H.-X. Li, J. Chu, and T.-J. Tarn, "Fidelity-based probabilistic Q-learning for control of quantum systems," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 5, pp. 920–933, 2014.

[22] D. Dong, C. Chen, J. Chu, and T.-J. Tarn, "Robust quantum-inspired reinforcement learning for robot navigation," *IEEE/ASME Transactions on Mechatronics*, vol. 17, no. 1, pp. 86–97, 2012.

[23] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, Mass, USA, 1998.

[24] T. Kollar and N. Roy, "Using reinforcement learning to improve exploration trajectories for error minimization," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '06)*, pp. 3338–3343, Orlando, Fla, USA, May 2006.

[25] H. H. Viet, P. H. Kyaw, and T. C. Chung, "Simulation-based evaluations of reinforcement learning algorithms for autonomous mobile robot path planning," in *IT Convergence and Services*, vol. 107 of *Lecture Notes in Electrical Engineering*, pp. 467–476, Springer, Amsterdam, The Netherlands, 2011.

[26] M. A. K. Jaradat, M. Al-Rousan, and L. Quadan, "Reinforcement based mobile robot navigation in dynamic environment," *Robotics and Computer-Integrated Manufacturing*, vol. 27, no. 1, pp. 135–149, 2011.

[27] M. Tan, "Multi-agent reinforcement learning: independent vs. cooperative agents," in *Proceedings of the 10th International Conference on Machine Learning (ICML '93)*, pp. 330–337, Amherst, Mass, USA, June 1993.

[28] L. Buşoniu, R. Babuška, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, vol. 38, no. 2, pp. 156–172, 2008.

[29] Y. Hu, Y. Gao, and B. An, "Multiagent reinforcement learning with unshared value functions," *IEEE Transactions on Cybernetics*, vol. 45, no. 4, pp. 647–662, 2015.