Research Article

A Hybrid Quantum Evolutionary Algorithm with Improved Decoding Scheme for a Robotic Flow Shop Scheduling Problem

Weidong Lei,¹ Hervé Manier,² Marié-Ange Manier,² and Xinping Wang¹

¹*School of Management, Xi'an University of Science and Technology, Xi'an 710054, China*

²*Université Bourgogne Franche-Comté, UTBM, OPERA, 90100 Belfort, France*

Correspondence should be addressed to Weidong Lei; leiweidong@xust.edu.cn

Received 8 October 2016; Accepted 28 February 2017; Published 16 April 2017

Academic Editor: Calogero Orlando

Copyright © 2017 Weidong Lei et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

We aim at solving the cyclic scheduling problem with a single robot and flexible processing times in a robotic flow shop, which is a well-known optimization problem in advanced manufacturing systems. The objective of the problem is to find an optimal robot move sequence such that the throughput rate is maximized. We propose a hybrid algorithm based on the Quantum-Inspired Evolutionary Algorithm (QEA) and genetic operators for solving the problem. The algorithm integrates three different decoding strategies to convert quantum individuals into robot move sequences. The Q-gate is applied to update the states of Q-bits in each individual. Besides, crossover and mutation operators with adaptive probabilities are used to increase the population diversity. A repairing procedure is proposed to deal with infeasible individuals. Comparison results on both benchmark and randomly generated instances demonstrate that the proposed algorithm is more effective in solving the studied problem in terms of solution quality and computational time.

1. Introduction

Since the 1970s, with the development of the robotics and automation technologies, computer-controlled robots instead of workers have been gradually used in many manufacturing industries to perform high frequency or dangerous transportation jobs. The advantages of robotic or automated manufacturing systems include higher production rate, better product quality, and more efficient use of materials, improved safety, and reduced labor intensity. Besides, highly robotic or automated manufacturing systems can effectively meet the requirement of mass production and respond to global competition.

In automated manufacturing systems, the production line with transporting device usually consists of several processing machines arranged in a line and one or more robots for transporting parts among machines, as shown in Figure 1. Due to the industrial applications [1], the part processing time on each machine is usually flexible and imposed with a pair of minimum and maximum time bounds. Besides, the cyclic production mode is usually adopted in such systems because of easy implementation in a mass production environment.

This leads to a repetitive schedule performed by the robot in every certain time. The duration of the robot to perform the repetitive schedule is called the cycle time or cycle length [2].

For such problem, scheduling of robot movements is very critical because the line productivity and the product quality extremely depend on it. Therefore, the decision concerns how to sequence the robot movements such that the cycle time is minimized, which is equivalent to maximizing the throughput rate of the production line in steady-state. Note that such scheduling problems in different industrial contexts may have different appellations, such as hoist scheduling problem [3], robotic cell, or robotic flow shop scheduling problem [4, 5].

Lei and Wang [6] first proved that the cyclic flow shop scheduling problem with a single robot and flexible processing times is NP-complete. Many researchers have been constantly dedicated to this area and proposed various efficient methods for solving the relevant problems. The first fundamental work in this area was done by Phillips and Unger [7], who formulated a Mixed Integer Programming (MIP) model for solving the single robot cyclic scheduling

FIGURE 1: A typical robotic production line.

problem with one degree, which means that each machine is emptied exactly once within a cycle. After that, various exact methods or heuristics were developed for the related problem in the literature [2, 8–10]. Moreover, Liu et al. [11] developed an MIP model for solving the problem with complex lines (i.e., including multifunction stations and parallel stations). Che and Chu [12] developed an efficient branch-and-bound (B&B) algorithm for the same problem as Liu et al. [11].

Besides the above studies, researchers have also examined the problem with multiple robots. For instance, Lei and Wang [13] and Zhou and Liu [14] proposed various heuristic algorithms based on zone-partition approach for solving different kinds of multiple robots cyclic scheduling problems. Manier et al. [15] first formulated a mathematical model with disjunctive constraints and then proposed a branch-and-bound like algorithm for finding the optimal schedules for multiple robots. Manier and Lamrous [16] proposed an evolutionary approach to minimize the number of robots on parallel tracks. Moreover, Che and Chu [17] and Leung et al. [18] presented a specific B&B algorithm and an MIP approach for the multiple robots scheduling problem with unidirectional part flow, where the part processing route is the same as the machine arrange sequence. Recently, Elmi and Topaloglu [19] proposed an MIP model for the multiple robots scheduling problem with multidegree.

Since a higher degree of cyclic schedule would generally improve the production rate, different MIP approaches were suggested for the single robot cyclic scheduling problem with 2 degrees, which means that each machine is emptied exactly 2 times during a cycle [20]. Feng et al. [21] proposed an MIP approach for the dynamic hoist scheduling problem with multiparts. Che et al. [22] proposed a B&B algorithm for the problem with k -degree cycles. More recently, Elmi and Topaloglu [23] developed a metaheuristic algorithm based on ant colony optimization for solving the robotic flow shop scheduling problem with multidegree. Furthermore, Li and Fung [24] proposed an MIP approach for the single robot multidegree cyclic scheduling with reentrance.

In recent years, Quantum-Inspired Evolutionary Algorithm (QEA) has been receiving much attention from researchers. It can be seen as a probability optimization algorithm based on the concepts and principles of quantum computation [25, 26]. Since 1990s, QEA has been successfully applied to solve several well-known optimization problems, such as travelling salesman problem [26], knapsack problems [27, 28], and lane reservation problem [29]. Particularly,

various variants of QEA have been proposed for solving different production scheduling problems. For instance, Li and Wang [30] employed QEA to solve the multiobjective flow shop scheduling problem. Zheng and Yamashiro [31] proposed different QEAs for the classic flow shop scheduling problem. Moreover, Gu et al. [32] proposed a parallel QEA for solving the stochastic job shop scheduling problem. Later on, Gu et al. [33] proposed a coevolutionary QEA for the same problem.

Due to the NP-completeness of the studied problem, the computation time spent by exact methods usually increases exponentially with its size. Thus, it is a wise choice to use metaheuristics to find sufficiently good schedules within a reasonable time. Because of its several unique advantages, such as better population diversity and very well global search ability, QEA has gained great success in many different optimization problems. Up to now, there is no work reported on using QEA to solve the robotic flow shop scheduling problem. So in this article, we propose a new scheduling algorithm based on QEA and genetic operators for the single robot cyclic scheduling problem with flexible processing times in a robotic flow shop.

A solution is usually represented by a permutation of job input sequence in classic flow shop or job shop scheduling problems. However, a chromosome is encoded by Q-bits in QEA, which is then converted into a binary chromosome. That is, QEA is generally based on a binary encoding. For this reason, a key issue in the development of QEA for production scheduling problems is to design an efficient decoding mechanism to convert a binary representation into a permutation-based representation. Typically, there are mainly two decoding schemes used in QEAs in the literature for solving various scheduling problems: binary-decimal decoding and shifting decoding [30]. For the binary-decimal decoding, it first uses a binary segment for each job and then converts it into a decimal number. After that, all jobs are sequenced based on their corresponding converted decimal numbers. It is understandable that the chromosome under such a scheme is usually very long, especially when the problem size is large. As a result, the search efficiency of the algorithm may be reduced. As for shifting decoding, it uses a permutation chromosome as a parent pattern and shifts its genes with the direction of a binary chromosome so as to generate a new permutation chromosome. Such a decoding usually has a better computational efficiency than binary-decimal decoding. But it cannot make full use of

the advantage of QEA due to its permutation-based representation.

To overcome the above drawbacks, we propose a new decoding scheme in this study. In our scheme, a binary chromosome is directly converted into permutation chromosome (i.e., a robot move sequence) using several different decoding procedures. Our decoding scheme can efficiently exploit the solution diversity due to Q-bits chromosome compared to shifting decoding and has a shorter chromosome than binary-decimal decoding [30].

The main contribution of this article is summarized as follows. Firstly, we propose a new decoding scheme as mentioned above. Secondly, we propose a more effective repairing procedure than the one in [10]. Note that, in [10], the reparation is conducted by randomly swapping any two moves for each infeasible sequence. In this article, we first identify the move segment that causes infeasibility of the entire move sequence and then repair it. Finally, to increase the population diversity, crossover and mutation operators with adaptive probabilities are also implanted into our algorithm.

The remainder of this article is organized as follows. In Section 2, we introduce the problem description and show an illustrative example of the problem as well as the problem formulation. The proposed algorithm with a repairing procedure is the subject of Section 3. The experimental results as well as comparisons of the proposed algorithm with the existing approach are given in Section 4. Last section gives some conclusions and perspectives.

2. Problem Statement and Mathematical Model

2.1. Problem Statement. For description consistency, we briefly give a problem description and notation, which are similar to those existing in the literature [2, 10, 12]. Given N processing machines (i.e., $M_1, M_2, \dots, M_N$) in a production line and a single robot for part transportation, machine 0 (i.e., M_0) and machine $N + 1$ (i.e., M_{N+1}) are the input station and the output station, respectively. After a part is unloaded from M_0 , it is to be successively processed through M_1 to M_N . The robot moving a part from M_i to M_{i+1} ($0 \leq i \leq N$) is called (loaded) move i , which includes three suboperations: (1) unloading a part from a machine; (2) carrying the part to the next machine; (3) loading the part into the machine. The robot movement without carrying a part between a pair of machines is called an empty move.

Moreover, the part processing time at each machine is flexible and should be confined within a pair of minimum and maximum time bounds. If the actual processing time violates the time limits, defective parts would be produced. Furthermore, at any time, each machine can process only one part. When a processing operation on a machine ends, the part must be moved by the robot to the next one without delay. The production lines run in a cyclic mode. In each cycle, each machine is emptied exactly once during a cycle, which involves cyclic schedules with one-degree. This article studies the one-degree cyclic scheduling problem with a

FIGURE 2: An example of cyclic schedule with a single robot.

single robot, and the decision concerns how to optimize the sequence of robot movements so as to minimize the cycle time.

For consistency of the problem formulation, we use the following notations and variables in this article, which are similar to [2, 12]:

$[a_i, b_i]$: the minimum and maximum bounds of the processing time on M_i , respectively, $1 \leq i \leq N$.

θ_i : the time needed to perform move i , $0 \leq i \leq N$.

$\delta_{i,j}$: the travel time for empty robot from M_i to M_j ; note that $\delta_{i,i} = 0$ and $\delta_{i,j} = \delta_{j,i}$, $0 \leq i, j \leq N + 1$. Note that $\delta_{i,j}$ satisfies the triangle inequalities; that is, $\delta_{i,j} \leq \delta_{i,k} + \delta_{k,j}$, $k \neq \{i, j\}$, $i \neq j$, $0 \leq i, j \leq N + 1$.

The decision variables are the following ones:

T : cycle time. It is the duration of a cycle.

S_i : the start time of (loaded) move i within a cycle, $0 \leq i \leq N$. Without loss of generality, move 0 is supposed to be the first move of a cycle; thus $S_0 = 0$.

To facilitate the formulation, we define the following intermediate variables:

c_i : if $c_i = 0$, then M_i is empty at the beginning of a cycle; else $c_i = 1$; then M_i is occupied by a part, $0 \leq i \leq N$. Define $C_N = \{c_0, c_1, \dots, c_N\}$. It is called the initial part distribution at the beginning of a cycle. Without loss of generality, we assume $c_0 = 1$ and $c_1 = 0$, since M_0 is always occupied by part and move 0 is the first move of a cycle.

$r[i]$: the $i + 1$ th move performed by the robot within a cycle, $0 \leq i \leq N$. As mentioned above, we have $r[0] = 0$. Define $R_N = \langle r[0], r[1], r[2], \dots, r[N] \rangle$, which represents the sequence of moves during a cycle. An example of R_N with $N = 3$ is $R_3 = \langle 0, 2, 3, 1 \rangle$, where $r[1] = 2$, $r[2] = 3$, and $r[3] = 1$, as shown in Figure 2. Here, $r[1] = 2$ means that the second move transfers a part from M_2 to M_3 .

Figure 2 shows an illustrative example of the studied problem with $N = 3$. In this example, there are three

processing machines (i.e., M_1 , M_2 , and M_3) with a single robot for part transportation as well as the loading station (i.e., M_0) and the unloading station (i.e., M_4). In Figure 2, the inclined solid arrows and the broken arrows represent the loaded moves and the empty moves, respectively. The start point and end point of an inclined solid arrow (resp., a broken arrow) represent the start time and the end time of corresponding loaded (resp., empty) move, respectively. Furthermore, the horizontal solid line represents the duration of the part processing operation. The production line is supposed to be in steady-state. As can be seen from Figure 2, at time 0, M_2 is the only machine to be occupied (and implicitly M_0). So the initial part distribution is $C_3 = \{1, 0, 1, 0\}$. For this distribution, the optimal robot move sequence is $R_3 = \langle 0, 2, 3, 1 \rangle$ (i.e., $S_0 < S_2 < S_3 < S_1$). When move 1 finishes, the robot comes back to M_0 and performs move 0 of the next cycle. We can also see that the robot performs the same loaded (or empty) move sequence in time interval $[T, 2T]$ as those ones in time interval $[0, T]$. This is called cyclic production mode. The duration of the repetitive sequence (i.e., R_3) is the cycle time T .

2.2. Mathematical Model. As mentioned above, move 0 is supposed to start at time 0; then the start times of other moves are all greater than 0. Thus, we have

$$\begin{aligned} S_0 &= 0, \\ S_i &> 0, \quad 1 \leq i \leq N. \end{aligned} \quad (1)$$

In Figure 2, we notice that the start time of processing operation i is the same as the end time of loaded move $i - 1$ (i.e., $S_{i-1} + \theta_{i-1}$); the end time of processing operation i is the same as the start time of loaded move i (i.e., S_i). Moreover, there are in total two possible states (empty or occupied) for each machine at the beginning of a cycle. Based on the above observations, the actual processing time on M_i can be represented as $S_i - (S_{i-1} + \theta_{i-1})$ and $T + S_i - (S_{i-1} + \theta_{i-1})$, for $c_i = 0$ and $c_i = 1$, respectively. Considering the processing time requirements, we have

$$a_i \leq c_i T + S_i - (S_{i-1} + \theta_{i-1}) \leq b_i, \quad 1 \leq i \leq N. \quad (2)$$

Furthermore, the robot must have enough time to perform any two successive moves (i.e., $r[i]$ and $r[i + 1]$); thus, the following relation holds:

$$S_{r[i]} + \theta_{r[i]} + \delta_{r[i]+1, r[i+1]} \leq S_{r[i+1]}, \quad 0 \leq i \leq N - 1. \quad (3)$$

When the last move (i.e., $r[N]$) finishes, the robot must come back to M_0 for executing move 0 of the next cycle. Hence, we have

$$S_{r[N]} + \theta_{r[N]} + \delta_{r[N]+1, 0} \leq T, \quad 1 \leq r[N] \leq N. \quad (4)$$

Based on the above works, the mathematical model for the single robot one-degree cyclic scheduling problem with flexible processing times can be formulated as the following [2]:

$$\begin{aligned} \min. \quad & T \\ \text{s.t.} \quad & (1)-(4). \end{aligned} \quad (5)$$

3. Hybrid Method

3.1. QEA Concepts. Quantum Evolutionary Algorithm (QEA) is formed based on the principles of quantum computation [25]. In a quantum computer, Q-bit is the smallest unit of information. Each Q-bit may be in “0” state, “1” state, or any superposition of the two. The following equation is usually used to define a Q-bit [27, 30]:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \quad \text{where } |\alpha|^2 + |\beta|^2 = 1. \quad (6)$$

In (6), α and β are two complex numbers, which represent the probability amplitudes of states 0 and 1, respectively. As a result, $|\alpha|^2$ and $|\beta|^2$ represent the probabilities that the Q-bit would be found in the “0” and “1” states, respectively. Suppose that a quantum individual is a string of N Q-bits [as shown below in (7)]; this individual can represent 2^N states at the same time. Such a representation commonly has a better population diversity compared with other representations [30]. However, each Q-bit finally collapses to a single state by using the way presented in Section 3.3, so QEA can be seen as a probabilistic algorithm. Besides, QEA uses Q-gate to change the values of α and β so as to influence the Q-bit state. Until now, several Q-gates have been proposed in the literature [30], such as the NOT gate, controlled NOT gate, and rotation gate.

QEA is like other evolutionary algorithms, such as genetic algorithm and annealing evolution algorithm, and it has a similar evolution paradigm. It begins with an initial population, in which each individual corresponds to a possible solution of a specific problem. After evaluating the population fitness, it applies Q-gate to update individuals for generating new offspring and then evaluates the new population. When the stop condition is satisfied, it ends and outputs the best solution. In what follows, we present in Section 3.2 the representation; in Section 3.3, we determine the states of Q-bits in each individual; in Section 3.4, we present the decoding scheme; in Sections 3.5 and 3.6, we describe the fitness evaluation function and the repairing procedure, respectively; in Section 3.7, we introduce the rotation gate and the genetic operators to update individuals; finally, in Section 3.8, we present the pseudocode of the proposed algorithm.

3.2. Representation. Indeed, we notice that machine state and Q-bit state have the same characteristics. That is, they both are either 0 or 1. Since precedence relations need to be determined between N moves in this article, we let Q-bit i corresponding to machine i , for $1 \leq i \leq N$, and use Rule 1 and Rule 2 introduced in the following section to determine each Q-bit state. If Q-bit i is in state “0” (i.e., $c_i = 0$), move $i - 1$ is performed before move i during a cycle; otherwise (i.e., $c_i = 1$), move i is performed before move $i - 1$ during a cycle. Hence, an individual Ψ containing N Q-bits is used to represent N machine states and is defined as follows:

$$\Psi = \begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_N \\ \beta_1 & \beta_2 & \cdots & \beta_N \end{bmatrix}, \quad (7)$$

where $|\alpha_i|^2 + |\beta_i|^2 = 1$, $1 \leq i \leq N$. Note that, in the initialization step, all Q-bits in Ψ are initialized as the equal probability (i.e., 1/2) of being 0 or 1. From above, we can know that each quantum individual corresponds to a complete part distribution C_N , more precisely the state of each M_i (i.e., empty or occupied).

In more classical and direct representations for the studied problem, each individual represents a moving sequence, so the value of gene j gives the index of the machine from which the j th move starts during one cycle. In such representations, the solution space contains $N!$ individuals. With our representation, we handle in a first step only 2^{N-1} individuals. This number may be further reduced for some instances with Rule 1, as explained in the following. Moreover, each Q-bits individual generally corresponds to several moving sequences, which we consider in a second step. Each time Rule 1 enables us to determine that an individual is not good, then all the associated moving sequences are unfeasible ones and there is no use evaluating them.

3.3. Initialization. For each specific instance, some machine states (or Q-bits states) may be directly determined by the following method. Specifically, we first suppose that $c_i = 1$; therefore move i occurs before move $i - 1$ within a cycle. Moreover, let us suppose that move $i - 1$ and move i are the last move and the second move of a cycle, respectively. Correspondingly, the minimum processing time of a part on machine M_i with $c_i = 1$ is $\delta_{i,0} + \theta_0 + \delta_{1,i}$. Indeed, move i would be the first move to be performed after move 0, and move $i - 1$ would be the last move of the cycle. Else, the processing time on M_i would be greater than $\delta_{i,0} + \theta_0 + \delta_{1,i}$, which would make the following assertion even more true. Then we can compare this processing time with b_i which is the maximum authorized time on M_i .

- (a1) If $b_i < \delta_{i,0} + \theta_0 + \delta_{1,i}$ (hereafter called Rule 1) happens, then we can know that the processing time requirement on M_i is violated. Consequently, all sequences relevant with $c_i = 1$ are infeasible ones. So c_i must be 0.
- (a2) Else, c_i may be 0 or 1.

Take move 1 and move 2 in Figure 2, for example, the processing time of a part on M_2 is equal to $\delta_{2,0} + \theta_0 + \delta_{1,2}$. Rule 1 can be used to reduce the enumerating space of C_N and thus improve the search efficiency. Indeed, if Rule 1 enables us to fix 0 to the values of p variables c_i , then the search space of C_N can be reduced to 2^{N-p-1} individuals.

For the state of Q-bit i in a quantum individual Ψ that is not determined by Rule 1, a random number rd_i is generated from the uniform distribution $[0, 1)$. If $rd_i > |\alpha_i|^2$, then Q-bit i is in state “1” (i.e., $c_i = 1$); else, Q-bit i is in state “0” (i.e., $c_i = 0$). This method is called Rule 2. Based on the above, the states of all Q-bits in one individual can be easily determined by Rule 1 and Rule 2, that is to say, the initial part distribution C_N .

3.4. Decoding Scheme. In what follows, we present how we derive the robot move sequence from a quantum individual.

```

Input:  $C_N = \{c_0, c_1, \dots, c_N\}$ 
Begin
  for  $i = 0$  to  $N$ 
     $\lambda_i \leftarrow c_i$ ;
   $r[0] \leftarrow 0$ ;  $\Phi \leftarrow \emptyset$ ;  $\Omega \leftarrow \emptyset$ ;
   $\lambda_1 \leftarrow 1$ ;  $\lambda_{N+1} \leftarrow 0$ ;  $\Phi \leftarrow \Phi \cup \{r[0]\}$ ;
  for  $i = 1$  to  $N$ 
    if ( $\lambda_i == 1 \& \lambda_{i+1} == 0$ ) then
       $\Omega \leftarrow \Omega \cup \{i\}$ ;
  for  $j = 1$  to  $N$ 
    find  $\max\{|\alpha_k|^2, k \in \Omega\}$ ;
     $r[j] \leftarrow k$ ,  $\Phi \leftarrow \Phi \cup \{k\}$ ,  $\lambda_k \leftarrow 0$ ;
    if ( $k! = N$ ) then  $\lambda_{k+1} \leftarrow 1$ ;
     $\Omega \leftarrow \emptyset$ ;
    for  $m = 1$  to  $N$ 
      if ( $\lambda_m == 1 \& \lambda_{m+1} == 0 \& m \notin \Phi$ )
        then  $\Omega \leftarrow \Omega \cup \{m\}$ ;
  End
Output:  $R_N = \{r[0], r[1], r[2], \dots, r[N]\}$ 

```

PSEUDOCODE 1: Pseudocode of Decoding Procedure 1.

For a better diversification, three different decoding procedures described in the following are used to convert a quantum individual into possible robot move sequence R_N , providing that the states of all Q-bits (i.e., C_N) in a quantum individual are already determined.

3.4.1. Decoding Procedure 1. For ease of description, we first define λ_i as a copy of c_i and $\lambda_i = c_i$. Let Φ be a set that records the performed moves. It should be note that λ_i can be seen as an indicator that indicates the state (i.e., empty or occupied) of M_i in the process. Thus, the value of λ_i is dynamically modified in the process. That is, when move i finishes, both the states of M_i and M_{i+1} are changed; that is, M_i becomes empty and M_{i+1} is occupied by a part. Thus, we set $\lambda_i = 0$, $\lambda_{i+1} = 1$ and put move i into set Φ .

Procedure 1 mainly depends on the probability sizes of Q-bits in Ψ to derive the move sequence. In particular, for each given C_N , when move $r[k]$ finishes, for $0 \leq k \leq N$, we first determine the possible candidates for next move $r[k+1]$; if move i satisfies the condition of $\lambda_i = 1$ and $\lambda_{i+1} = 0$ (note that if $i = N$, the output station can be seen as always empty) and $i \notin \Phi$, then we set $\Omega = \Omega \cup \{i\}$. Notice that we use set Ω to store all possible candidates for the next move $r[k+1]$. Finally, we choose move j with the highest probability (i.e., $|\alpha_j|^2$) in set Ω as move $r[k+1]$, and let $\lambda_j = 0$, $\lambda_{j+1} = 1$ (for $j \neq N$), $\Phi = \Phi \cup \{j\}$, and $\Omega = \emptyset$. In the next step, we update all related parameters and use a similar way to derive the following move (i.e., $r[k+2]$). Once the whole sequence (i.e., R_N) is determined, this procedure stops. Pseudocode 1 illustrated the process of Decoding Procedure 1.

For example, a complete part distribution (corresponding to a quantum individual) C_N with $N = 5$ is $C_5 = \{1, 0, 1, 0, 1, 0\}$. When the first move (i.e., $r[0]$) finishes, by using the above decoding procedure, we have $\lambda_1 = 1$, $\lambda_2 = 1$, $\lambda_3 = 0$, $\lambda_4 = 1$, $\lambda_5 = 0$, and $\Phi = \{0\}$, from which we can know that

M_1, M_2 , and M_4 are currently occupied by a part. As the robot cannot unload a part from an empty machine and also cannot load a part into an occupied machine, we have $\Omega = \{2, 4\}$. Finally, according to the selection rule, if $|\alpha_2|^2 \geq |\alpha_4|^2$, we have $r[1] = 2$; else, $r[1] = 4$. The similar ways are used to update λ_i, Φ, Ω and then determine $r[k], 2 \leq k \leq 5$.

3.4.2. Decoding Procedure 2. For ease of description, we keep the intermediate parameters λ_i, Φ , and Ω defined in Procedure 1. Furthermore, we define t_i as the start time of move i in the process of deriving the whole sequence, for $0 \leq i \leq N$, and let $t_0 = 0$. To derive a move sequence from each given C_N , Procedure 2 mainly depends on the rule of minimal time unit increment between $t_{r[k]}$ and $t_{r[k+1]}$, for $1 \leq k \leq N$, while respecting the processing time requirement, since the objective of the problem is to minimize the cycle time T . That is to say, in each step, we have a set of several moves and choose one move with the earliest starting time as move $r[k+1]$ from the set.

In particular, on one hand, when move $r[k]$ finishes, as similarly done in Procedure 1, we derive the value of Ω from each given C_N . On the other hand, we design a different strategy to determine move $r[k+1]$ compared with the Procedure 1. At first, we calculate each t_j in set Ω ; that is, $t_j = t_{r[i]} + \theta_{r[i]} + \delta_{r[i+1],j}$. Then, for each move j in set Ω , we check whether move $j-1$ exists in the partial sequence $\langle r[0], r[1], \dots, r[i] \rangle$. If it exists and $t_j - t_{j-1} - \theta_{j-1} < a_j$ happens, then we update $t_j = t_{j-1} + \theta_{j-1} + a_j$ so as to meet the processing requirement. Finally, we choose move j with the smallest value of t_j in set Ω as move $r[k+1]$, and let $\lambda_j = 0$, $\lambda_{j+1} = 1$ (for $j \neq N$), $\Phi = \Phi \cup \{j\}$, and $\Omega = \emptyset$. In the next step, we update all related parameters so as to derive move $r[k+2]$. Once the whole sequence (i.e., R_N) is determined, this procedure stops. Pseudocode 2 illustrated the process of Decoding Procedure 2.

For instance, an example of C_N with $N = 5$ is $C_5 = \{1, 0, 0, 1, 0, 1\}$. When the first move (i.e., $r[0]$) finishes, by definitions, we have $\lambda_1 = 1, \lambda_2 = 0, \lambda_3 = 1, \lambda_4 = 0, \lambda_5 = 1$, and $\Phi = \{0\}$ as well as $\Omega = \{1, 3, 5\}$. We first calculate t_1 (note that if $t_1 - \theta_0 < a_1$, then we set $t_1 = \theta_0 + a_1$) and then t_3 and t_5 by $t_0 + \theta_0$ plus $\delta_{1,1}, \delta_{1,3}, \delta_{1,5}$, respectively, and then choose the move with the smallest starting time among the three candidates as $r[1]$. The similar ways are used to update λ_i, Φ, Ω and then determine $r[k], 2 \leq k \leq 5$.

3.4.3. Decoding Procedure 3. Procedure 3 mainly depends on the precedence relationship between move $i-1$ and move i (i.e., the value of c_i) to derive the move sequence. For each given C_N (i.e., a quantum individual), if $c_i = 1$, then move i is set before move $i-1$ in R_N ; else, move i is set after move $i-1$ in R_N . Pseudocode 3 illustrated the process of Decoding Procedure 3.

For instance, examples of C_N and R_N with $N = 5$ are, respectively, $C_5 = \{1, 0, 1, 1, 0, 1\}$ and $R_5 = \langle 0, 2, 1, 4, 3, 5 \rangle$, from which we can easily derive a new sequence; that is, $R_5 = \langle 0, 3, 2, 1, 5, 4 \rangle$. Note that, at the initial step, we set $r[i] = i, 0 \leq i \leq N$.

Based on the above descriptions, we first apply the three proposed decoding procedures to each quantum individual

```

Input:  $C_N = \{c_0, c_1, \dots, c_N\}$ 
Begin
   $r[0] \leftarrow 0; \Phi \leftarrow \emptyset; \Omega \leftarrow \emptyset; t_0 \leftarrow 0;$ 
  for  $i = 0$  to  $N$ 
     $\lambda_i \leftarrow c_i;$ 
     $\lambda_1 \leftarrow 1; \lambda_{N+1} \leftarrow 0; \Phi \leftarrow \Phi \cup \{r[0]\};$ 
  for  $i = 1$  to  $N$ 
    if  $(\lambda_i == 1 \& \& \lambda_{i+1} == 0)$  then
       $\Omega \leftarrow \Omega \cup \{i\};$ 
  for  $i = 1$  to  $N$ 
    compute  $t_j = t_{r[i]} + \theta_{r[i]} + \delta_{r[i+1],j}, j \in \Omega;$ 
    for  $m = 0$  to  $i-1, j \in \Omega$ 
      if  $((j - r[m]) == 1 \& \& t_j - t_{j-1} - \theta_{j-1} < a_j)$ 
        then  $t_j = t_{j-1} + \theta_{j-1} + a_j;$ 
    find min  $t_j, j \in \Omega;$ 
     $r[i] \leftarrow j, \Phi \leftarrow \Phi \cup \{j\}, \lambda_j \leftarrow 0;$ 
    if  $(j \neq N)$  then  $\lambda_{j+1} \leftarrow 1;$ 
     $\Omega \leftarrow \emptyset;$ 
    for  $m = 1$  to  $N$ 
      if  $(\lambda_m == 1 \& \& \lambda_{m+1} == 0 \& \& m \notin \Phi)$ 
        then  $\Omega \leftarrow \Omega \cup \{m\};$ 
End
Output:  $R_N = \langle r[0], r[1], r[2], \dots, r[N] \rangle$ 

```

PSEUDOCODE 2: Pseudocode of Decoding Procedure 2.

and then select the best sequence (i.e., the best fitness) from the three generated sequences to represent this individual.

3.5. Fitness Evaluation. To facilitate the description, $\text{fit}(X)$ is defined to represent the fitness value of each individual X . Basically the higher the fitness of an individual (i.e., $\text{fit}(X)$), the better the quality of individual X . Therefore, $\text{fit}(X)$ is computed as follows: $\text{fit}(X) = F/T$, where F is a positive parameter and set as 1 (note that in fact F can also be set as other positive integer values, and its value does not impact the result or the procedure running) in this article. From this definition, we see that the smaller the cycle time T ($T > 0$), the greater the fitness value. For each individual relevant with a robot move sequence, it is evaluated by using the graph-based polynomial algorithm developed by Chen et al. [2]. In particular, if the sequence is proved to be feasible, then the algorithm returns a positive value for the cycle time T and the individual fitness can be calculated; otherwise, the individual fitness is set to be 0.

3.6. Repairing Procedure. It should be noted that constraints (2)~(4) formulated in Section 2.2 can be regarded as two classes. One is flexible processing time constraints and the other is robot transportation capacity constraints, which are (2), (3), and (4), respectively. Generally, if a sequence R_N is infeasible, the following cases happen:

- (C1) the flexible processing time constraint is violated.
- (C2) the robot transportation capacity constraint is violated.

```

Input:  $C_N = \{c_0, c_1, \dots, c_N\}$ ,  $R_N = \langle r[0], r[1], \dots, r[N] \rangle$ 
Begin
  for  $i = 1$  to  $N - 1$ 
     $p \leftarrow 0, q \leftarrow 0, k \leftarrow 0, l \leftarrow 0, m \leftarrow 0$ ;
    for  $j = 1$  to  $N$ 
      if  $(r[j] == i)$  then  $p \leftarrow j, l \leftarrow l + 1$ ;
      if  $(r[j] == i + 1)$  then  $q \leftarrow j, m \leftarrow m + 1$ ;
      else if  $(l! = 0 \& \& m! = 0)$  break;
    if  $(c_{i+1} == 0)$  then
      if  $(p > q)$  then
         $k \leftarrow r[q]$ , and for  $j = q$  to  $N - 1$ 
           $r[j] \leftarrow r[j + 1], r[N] \leftarrow k$ ;
      else
        if  $(p < q)$  then
           $k \leftarrow r[q]$ , and for  $j = q$  to  $2$ 
             $r[j] \leftarrow r[j - 1], r[1] \leftarrow k$ ;
    End
Output:  $R_N = \langle r[0], r[1], r[2], \dots, r[N] \rangle$ 

```

PSEUDOCODE 3: Pseudocode of Decoding Procedure 3.

In what follows, we present the repairing procedure based on the above cases. For an individual with an associated robot move sequence R_N , we identify each partial sequence in a whole robot move sequence R_N which is either in sequence of $i - 1 \rightarrow \bullet \rightarrow \bullet \rightarrow i$ (which means move $i - 1$ is performed before move i within a cycle) or of $i \rightarrow \bullet \rightarrow \bullet \rightarrow i - 1$ (which means move i is performed before move $i - 1$ within a cycle). That is to say, a complete robot sequence R_N consists of N pieces of such a partial sequence. For ease of description, we define the following parameters:

$z_{i-1,i}$: the duration between the finish time of move $i - 1$ and the start time of move i for a partial sequence $i - 1 \rightarrow \bullet \rightarrow \bullet \rightarrow i$, for $1 \leq i \leq N$. Note that $z_{i-1,i}$ generally is equal to the sum of all loaded move (denoted by $\bullet$) times and relevant empty move times. If there exists a pair of moves $j - 1$ and j in the sequence $i - 1 \rightarrow \bullet \rightarrow j - 1 \rightarrow \bullet \rightarrow j \rightarrow i$ and $z_{j-1,j} < a_j$, then we let $z_{i-1,i} = z_{i-1,i} + a_j - z_{j-1,j}$. Note that $z_{i-1,i}$ may span the cycle or be within a cycle. For example, in Figure 2, the two consecutive sequences are $0 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 0 \rightarrow 2 \rightarrow 3 \rightarrow 1$. From it, we can see that $z_{0,1}$ and $z_{2,3}$ are within a cycle, but $z_{1,2}$ spans the cycle. Therefore, $z_{i-1,i}$ can be used to check the satisfaction of flexible processing requirements no matter $c_i = 0$ or $c_i = 1$.

d : the mean time of all loaded move times, $d = \sum_{i=0}^N \theta_i / (N + 1)$.

e : the mean time of all empty move times, $e = 2(\sum_{i=0}^{N+1} \sum_{j=i}^{N+1} \delta_{i,j} - \delta_{0,1}) / (N^2 + 5N)$. Note that the possible number of empty moves is $(N^2 + 5N + 6)/2$. Since the empty moves between M_0 and M_0 , M_0 and M_1 , and M_{N+1} and M_{N+1} do not actually happen, the number is reduced to $(N^2 + 5N)/2$.

For an infeasible sequence R_N , we first use parameters $z_{i-1,i}$ to check the sequence R_N .

- (1) If $z_{i-1,i}$ is verified to be greater than its upper bound b_i , then we remove one or more moves from the corresponding partial sequence, so as to make the partial sequence feasible; else if $z_{i-1,i}$ is verified to be smaller than its lower bound a_i and the time gap between a_i and $z_{i-1,i}$ is greater than the sum of d and $2e$, then we insert possible moves into the partial sequence.
- (2) Then, we identify the violated robot capacity constraints by the start times of all moves (i.e., S_i , $1 \leq i \leq N$) given by the evaluation process. For ease of description, let moves i and j be the identified two moves violating the robot capacity constraints; that is, $S_i + \theta_i + \delta_{i+1,j} > S_j$, with $S_i < S_j$. If these two moves are two consecutive moves, we set move j before move i in sequence R_N so as to make the sequence be feasible; else, we remove one or more moves between moves i and j so as to make the two moves satisfy the robot capacity constraints.

3.7. Update Individuals

3.7.1. Rotation Gate. In this paper, the rotation gate $U(\Delta\omega_i)$ is adopted as the variation operator to update the Q-bits in (7). ω_0 is set to be as the initial rotation angle. For individual X , the Q-bit i in it can be updated as the following way [27, 30]:

$$\begin{bmatrix} \alpha'_i \\ \beta'_i \end{bmatrix} = \begin{bmatrix} \cos \Delta\omega_i & -\sin \Delta\omega_i \\ \sin \Delta\omega_i & \cos \Delta\omega_i \end{bmatrix} \begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix}. \quad (8)$$

We define fit_b as the fitness of the best individual found in population. The rotation angle $\Delta\omega_i$ is defined according to the respective values of the corresponding parameter c_i in the individual X (labeled c_{i-X}) and in the best one (labeled $c_{i-\text{best}}$). If the condition $\text{fit}(X) < \text{fit}_b$ holds, then consider the following conditions [27].

Case 1. If bit i is in the first or the third quadrant, then consider the following.

Case 1.1. If $c_{i-\text{best}} = 1$ and $c_{i-X} = 0$, then $\Delta\omega_i = -\omega_0$; here the rotation angle $\Delta\omega_i$ is set negative so as to increase the probability that Q-bit i is in state “1.”

Case 1.2. If $c_{i-\text{best}} = 0$ and $c_{i-X} = 1$, then $\Delta\omega_i = \omega_0$, and the rotation angle $\Delta\omega_i$ is set as positive so as to increase the probability that Q-bit i is in state “0.”

Case 1.3. Else, $\Delta\omega_i = 0$.

Case 2. If bit i is in the second or the fourth quadrant, then consider the following.

Case 2.1. If $c_{i-\text{best}} = 1$ and $c_{i-X} = 0$, then $\Delta\omega_i = \omega_0$; here the rotation angle $\Delta\omega_i$ is set as positive so as to increase the probability that Q-bit i is in state “1.”

Case 2.2. If $c_{i-\text{best}} = 0$ and $c_{i-X} = 1$, then $\Delta\omega_i = -\omega_0$, and the rotation angle $\Delta\omega_i$ is set as negative so as to increase the probability that Q-bit i is in state “0.”

FIGURE 3: Crossover and mutation operators.

Case 2.3. Else, $\Delta\omega_i = 0$;

Besides, since the probability of a Q-bit i in state “0” may be equal to 1 or 0, the updated Q-bit i may be trapped in state “0” or “1,” which may lead to the premature convergence of population. Thus, a small constant μ is applied to ensure that the probabilities of the two states both belonged to the range $[\mu, 1 - \mu]$. As a result, the following equation must be considered [34]:

$$\begin{bmatrix} \alpha'_i \\ \beta'_i \end{bmatrix} = \begin{cases} [\sqrt{\mu}, \sqrt{1-\mu}]^T, & \text{if } \alpha'_i < \sqrt{\mu}, \\ [\sqrt{1-\mu}, \sqrt{\mu}]^T, & \text{if } \alpha'_i > \sqrt{\mu}, \\ [\alpha' \ \beta']^T, & \text{else.} \end{cases} \quad (9)$$

By applying the decoding procedures given in Section 3.4 to each updated quantum individual, robot move sequences can be generated from it.

3.7.2. Genetic Operators. In this subsection, selection, crossover, and mutation operators are applied to further evolve the population [35]. To facilitate the description, the following notations are given:

P_c, P_m : crossover and mutation probabilities, respectively.

fit_a: the average fitness of the entire population.

fit₀: the maximum fitness of a specific instance, which is computed as follows: $\text{fit}_0 = 1/T_L$. T_L is the lower bound on cycle time T for the instance. It can be obtained by the following way, which is taken from Chen et al. [2]:

$$T_L \geq \max(a_i + \theta_i + \theta_{i-1} + \delta_{i+1,i-1}), \quad 1 \leq i \leq N. \quad (10)$$

According to Srinivas and Patnaik [36], P_c and P_m are defined, respectively, in a similar way:

$$\begin{aligned} P_c &= 0.7 \times \frac{[\text{fit}_0 - \text{fit}_b]}{[\text{fit}_0 - \text{fit}_a]}, \\ P_m &= 0.5 \times \frac{[\text{fit}_0 - \text{fit}(X)]}{[\text{fit}_0 - \text{fit}_a]}. \end{aligned} \quad (11)$$

Adaptively adjusting P_c and P_m (i.e., (11)) can prevent divergence and escape from the local optimal, since (11)

can dynamically reduce P_c and P_m for individuals with high fitness or increase P_c and P_m for individuals with low fitness.

In this research, two-point crossover operator is applied to generate the offspring. First, two individuals are chosen by the binary tournament method as parents 1 and 2; then, for parent 1, two different positions p and q are randomly chosen ($p < q$), $p, q \in [1, N]$. For $i \in [1, p]$ and $(q, N]$, the values of $r[i]$ for the new offspring 1 are inherited from parent 1. For $i \in [p, q]$, the new $r[i]$ is sequentially chosen from parent 2, on condition that its value was not already chosen from parent 1. The same operations are done, starting with parent 2 and then parent 1, to generate offspring 2. This operation is depicted in Figure 3(a), in which $|$ is the chosen position.

Besides, a mutation operator is adopted to prevent a solution falling into a local optimum, which is designed as follows. For a chosen individual $R_N = \langle r[0], r[1], r[2], \dots, r[N] \rangle$, first, we randomly choose a position p , $p \in [1, N]$ and then reorder the partial move sequence $\langle r[p+1], r[p+2], \dots, r[N] \rangle$. This operation is depicted in Figure 3(b), in which $|$ is the chosen position.

3.8. Pseudocode and Complexity Analysis of the Proposed HQEA

3.8.1. Pseudocode of the Proposed HQEA. Based on the above works presented in Sections 3.1–3.7, the pseudocode of HQEA for solving the considered problem can be written as follows.

Input:

- Popsiz (population size).
- MaxIter (maximum number of generations).
- ω_0 (initial rotation angle).
- Rt (maximum repairing times).
- N (problem size)

Output: the best cycle time T and its related robot move sequence R_N .

Begin

- (a) $t \leftarrow 0$;
- (b) Initialize Q-bits chromosomes;
- (c) Determine Q-bit state (i.e., C_N) by Rule 1 and Rule 2;

- (d) Apply decoding scheme to generate possible move sequences (i.e., R_N);
- (e) Evaluate all obtained sequences and repair the infeasible ones;
- (f) Assign fitness to individuals;
- (g) Sort all individuals based on their fitness from best to worse;
- (h) Store the best solution at generation t ;

While ($t < \text{MaxIter}$) **do**

Begin

- (i) $t \leftarrow t + 1$;
- (j) Update each Q-bits chromosome by using quantum rotation gate;
- (k) Determine Q-bit state (i.e., C_N) by Rule 1 and Rule 2;
- (l) Apply decoding scheme to generate possible move sequences (i.e., R_N);
- (m) Evaluate all obtained sequences and repair the infeasible ones;
- (n) Assign fitness to individuals;
- (o) Sort all individuals based on their fitness from best to worse;
- (p) Store the best solution at generation t ;
- (q) Apply the proposed adaptive genetic operators to the obtained sequences;
- (r) Evaluate the obtained sequences and repair the infeasible ones;
- (s) Assign fitness to individuals;
- (t) Sort individuals and update the best solution;

End

If the stopping condition is reached, then the algorithm output the best solution, including cycle time T and its related robot move sequence R_N .

End

3.8.2. Complexity Analysis of the Proposed HQEA. The computational complexity of the proposed HQEA is analyzed as follows. Step (a) and Step (h), respectively, take $O(1)$ and $O(N)$ times. In step (b), initializing the entire population takes $O(2\text{Popsi}zeN)$ time. In step (c), determining the Q-bit state for all quantum individuals takes $O(\text{Popsi}ze(N - 1))$ time. In step (d), decoding all the quantum individuals takes $O(3\text{Popsi}zeN(N - 1))$ time. According to [2], evaluating all the individuals in step (e) takes $O(27N^6\text{Popsi}ze)$ time. Besides, the repairing procedure in step (e) takes $O(3N^2(N - 1)Rt\text{Popsi}ze)$ time. The fitness assignment operation in step (f) takes $O(3\text{Popsi}ze)$ time. Sequencing all the individuals in step (g) takes $O(\text{Popsi}ze \log \text{Popsi}ze)$ time. Consequently, the computational complexity for step (a)–step (h) is approximately $O(27N^6\text{Popsi}ze)$.

In the *while* loop, updating all the quantum individuals in step (j) takes $O(\text{Popsi}zeN)$ time. Step (q) takes $O(2\text{Popsi}zeN)$ time. Similar to step (a)–step (h), the computational complexity for other steps in the *while* loop is approximately

$O(36N^6\text{Popsi}ze)$. Since the *while* loop executes *MaxIter* times, the computational complexity for the *while* loop is about $O(36N^6\text{Popsi}ze\text{MaxIter})$. Then, the computational complexity is approximately $O(36N^6\text{Popsi}ze\text{MaxIter}) + O(27N^6\text{Popsi}ze)$, that is, $O(9N^6\text{Popsi}ze(4\text{MaxIter} + 3))$.

4. Experimental Results

To verify the applicability of the proposed algorithm, both benchmark and randomly generated instances were used in the experimental studies. All computational experiments were conducted on an ASUS Laptop with an Intel Core i5-3210M Processor 2.50 GHZ and on a Windows 8 environment. The parameters were set as follows: population size: $\text{Popsi}ze = 50$; the maximum number of generations: $\text{MaxIter} = 200$; initial rotation angle $\omega_0 = 0.05\pi$; $\mu = 0.008$. The maximum repairing times: $Rt = 6$. For evaluating the quality of the solution obtained with our HQEA, the problem was also formulated by the MIP approach and solved by the ILOG CPLEX (Version 12.4).

4.1. Benchmark Instances. The proposed algorithm was verified by using five well-known benchmark instances in the literature: Mini Phillips (Mini, $N = 8$), Black and Oxide 2 (BO2, $N = 11$), Phillips and Unger (P&U, $N = 12$), and Ligne 1 ($N = 12$) and Ligne 2 ($N = 14$), which can be found in [7, 18, 37–39].

Table 1 gives the experimental results for five benchmark instances obtained with our algorithm and CPLEX software, in terms of the number of remaining possible C_N after applying Rule 1 (Nb. C_N after Rule 1 for short), the convergence generation (Con.gen. for short), the best cycle times, and the CPU times (measured in seconds). The “Con.gen.” refers to how many generations are needed for our algorithm to find the best solution and no improvement on the solution in the later iterations. Consequently, the subcolumn “Con. time” represents the time needed by the “Con.gen.” and is computed as: $\text{Con.time} = \text{Con.gen.} \times (\text{Our'sCPUtime}/\text{MaxIter})$.

In Table 1, we can see that Rule 1 works well on two benchmark instances (i.e., Mini and P&U) as shown in column “Nb. C_N after Rule 1,” as the enumerating space of C_N is, respectively, reduced 50% for the two instances (note that there are in total 2^{N-1} individuals for each instance with given value of N). In column “best cycle times,” our algorithm finds the same solutions as the optimal ones obtained with the CPLEX (see “Our” and “CPLEX”), except for Ligne 1. The standard deviation (SD) of the best cycle time obtained with our algorithm from the optimal cycle time obtained with CPLEX for Ligne 1 is less than 5%. Although the CPU times spent by our algorithm are generally longer than those spent by CPLEX, we can also see in column “Con.gen.” that our algorithm finds the optimal solutions for most cases in very early generations. Note that the time gaps between Con. time and CPLEX are very narrow, less than 1 s. Due to this very small amount of gaps, the difference in CPU times between CPPEX and our algorithm is meaningless and can be negligible. In summary, our algorithm is an effective method

TABLE 1: Results for the benchmark instances.

Instances	Nb. C_N after Rule 1	Con.gen.	Best cycle times			CPU times (In seconds)		
			Our	CPLEX	SD	Our	Con. time	CPLEX
Mini	2^6	2	287	287	0	4.75	0.048	0.16
BO2	2^{10}	13	279.3	279.3	0	5.26	0.342	0.25
P&U	2^{10}	29	521	521	0	5.65	0.819	0.47
Ligne 1	2^{11}	24	411	392	4.84%	7.35	0.882	0.72
Ligne 2	2^{13}	26	712	712	0	6.71	0.872	0.48

TABLE 2: Results for the remaining number of C_N for each instance after applying Rule 1.

N	Group 1					Group 2				
	1	2	3	4	5	1	2	3	4	5
10	2^9	2^8	2^9	2^9	2^9	2^9	2^9	2^9	2^9	2^9
15	2^{14}	2^{14}	2^{14}	2^{13}	2^{14}	2^{14}	2^{14}	2^{14}	2^{14}	2^{14}
18	2^{16}	2^{17}	2^{17}	2^{16}	2^{17}	2^{16}	2^{17}	2^{17}	2^{16}	2^{15}
20	2^{18}	2^{19}	2^{19}	2^{17}	2^{18}	2^{18}	2^{18}	2^{18}	2^{18}	2^{19}
22	2^{18}	2^{21}	2^{20}	2^{20}	2^{18}	2^{19}	2^{20}	2^{21}	2^{19}	2^{19}

TABLE 3: Comparison results between our decoding scheme and shifting decoding scheme on Group 1 and Group 2.

N	Group 1					Group 2				
	Average cycle times			Average CPU times		Average cycle times			Average CPU times	
	Our	SQEA	AD	Our	SQEA	Our	SQEA	AD	Our	SQEA
10	400.4	401.2	-0.20%	6.74	10.12	318.4	318.4	0	6.83	17.8
15	607.2	628	-3.31%	24.56	51.79	470.6	470.8	-0.04%	37.44	146.68
18	808.8	817.4	-1.05%	54.88	286.55	627.4	638.2	-1.69%	49.46	267.57
20	897.2	927.8	-3.30%	117.53	360.14	678.6	690.2	-1.68%	141.02	275.59
22	1058.6	1351.2	-21.65%	274.43	315.62	802.6	878.2	-8.61%	190.16	373.43

for solving the benchmark instances in terms of solution quality and CPU times.

4.2. Random Instances. In this subsection, random instances are generated to further test the performance of the proposed algorithm. We compare our algorithm with the QEA with shifting decoding scheme to demonstrate the effectiveness of our decoding scheme. We also compare it with commercial software CPLEX and Tabu search (TS) algorithm [10]. The random instances are generated as follows. We set N which belongs to $\{10, 15, 18, 20, 22\}$, and let $U(c1, c2)$ be a uniform distribution between parameters $c1$ and $c2$. The random tests were set as two different groups. One (called Group 1) was defined as the following way: the time windows were $a_i = U(30, 120)$ and $b_i = a_i + U(10, 750)$, $1 \leq i \leq N$; the empty moves and the loaded moves were, respectively, defined as $\delta_{i,j+1} = U(3, 6)$, $\delta_{i,j} = \sum_{k=i}^{j-1} \delta_{k,k+1}$, $0 \leq i, j \leq N+1$, and $\theta_i = 20 + \delta_{i,i+1}$, $0 \leq i \leq N$. The other (called Group 2) was defined as $a_i = U(40, 120)$, $b_i = 30 + U(1, 8) \times a_i$, $1 \leq i \leq N$, $\delta_{i,i+1} = U(2, 5)$, $\delta_{i,j} = \sum_{k=i}^{j-1} \delta_{k,k+1}$, $0 \leq i, j \leq N+1$, $\theta_i = 15 + \delta_{i,i+1}$, and $0 \leq i \leq N$. These defined parameters were based on the magnitude of the data from real production lines [7, 38]. For each given N , 5 instances were randomly generated.

Table 2 reports the remaining number of C_N for each randomly generated instance after applying Rule 1. As mentioned before, there are in total 2^{N-1} individuals for each instance with a given value of N . As presented in Table 2, Rule 1 is efficient on 22 random instances (i.e., the numbers in bold font). We can also see in Table 2 that the enumerating space of C_N for each instance among the 22 instances is reduced at least 50% and at most 87.5% after applying Rule 1. Based on these results, Rule 1 seems efficient for the studied problem.

Firstly, we compare our algorithm with the QEA with shifting decoding scheme (i.e., SQEA). Table 3 presents the comparison results between our decoding scheme and shifting decoding scheme on Group 1 and Group 2. We can see that our decoding scheme generally outperforms the shifting decoding scheme in terms of solution quality and CPU times for all random instances. In particular, the deviations (i.e., AD) of our algorithm from that with shifting decoding generally decrease with the problem size. Besides, our algorithm spent less time than that with shifting decoding for all random instances.

Secondly, Tables 4 and 5 report the comparison results for randomly generated instances using our algorithm, Yan's algorithm, and commercial optimization software CPLEX. Columns AD¹ and AD², respectively, represent the standard

TABLE 4: Comparison results for the randomly generated instances Group 1.

N	Average cycle times					Average CPU times (In seconds)		
	Our	Yan	CPLEX	AD ¹	AD ²	Our	Yan	CPLEX
10	400.4	400.4	400.4	0	0	6.74	2.7	1.44
15	607.2	624.6	602.4	0.8%	-2.79%	24.56	19.95	42.95
18	808.8	859.4	797.6	1.4%	-5.89%	54.88	32.16	1351.53
20	897.2	914.6	865.8	3.63%	-1.90%	117.53	114.51	1692.12
22	1058.6	1122.4	1025	3.28%	-5.68%	274.43	211.34	2712.38

TABLE 5: Comparison results for the randomly generated instances Group 2.

N	Average cycle times					Average CPU times (In seconds)		
	Our	Yan	CPLEX	AD ¹	AD ²	Our	Yan	CPLEX
10	318.4	318.4	318.4	0	0	6.83	4.58	1.38
15	470.6	472.6	466.4	0.9%	-0.42%	37.44	62.35	51.50
18	627.4	636.4	612.6	2.42%	-1.41%	49.46	92.84	324.24
20	678.6	684	661.8	2.54%	-0.79%	141.02	53.52	1077.9
22	802.6	835.4	779.8	2.92%	-3.93%	190.16	102.62	1897.76

TABLE 6: Comparison results of our proposed algorithm with and without repairing procedure on randomly generated instances Group 1 and Group 2.

N	Group 1					Group 2				
	Average cycle times			Average CPU times		Average cycle times			Average CPU times	
	Our	Our [#]	AD	Our	Our [#]	Our	Our [#]	AD	Our	Our [#]
10	400.4	406.8	-1.57%	6.74	3.94	318.4	320	-0.50%	6.83	4.09
15	607.2	642.6	-5.51%	24.56	9.86	470.6	518.4	-9.22%	37.44	10.77
18	808.8	966.8	-16.34%	54.88	17.33	627.4	722.2	-13.07%	49.46	17.85
20	897.2	1058.2	-15.21%	117.53	36.31	678.6	1095.8	-38.07%	141.02	33.27
22	1058.6	1400	-24.39%	274.43	87.25	802.6	1199.2	-33.07%	190.16	63.25

Note. Our[#] represents our algorithm without repairing procedure; AD = (Our - Our[#])/Our[#] × 100%.

deviation of our solution from those obtained with CPLEX and Yan's algorithm; that is, $AD^1 = (Our - CPLEX)/CPLEX \times 100\%$ and $AD^2 = (Our - Yan)/Yan \times 100\%$. As presented in Tables 4 and 5, our algorithm and Yan's algorithm find the same solutions as the optimal ones obtained with CPLEX for random instances with $N = 10$. For the remaining random instances, the average cycle times obtained with our algorithm are smaller than those obtained with Yan's algorithm. As a result, the deviations (i.e., AD^2) of our algorithm from Yan's algorithm are all negative, which range from -5.89% to -1.9% in Table 4 and from -3.93% to -0.42% in Table 5. Note that the smaller the AD^2 , the better the solution quality our algorithm obtained over Yan's algorithm. Therefore, our algorithm has a better solution quality than Yan's algorithm. We also notice that CPLEX has a better solution quality than our algorithm and Yan's algorithm but it spent much longer CPU times, which will be discussed later. Moreover, the values of AD^1 in Tables 4 and 5 both increase with the problem size but are less than 4% and 3%, respectively, which are generally small and acceptable.

For the average CPU times, we can see from Tables 4 and 5 that both our algorithm and Yan's algorithm perform

much better than CPLEX for each value of N , except for $N = 10$. We also notice that Yan's algorithm has a better performance than our algorithm in terms of CPU times except for $N = 15$ and $N = 18$ in Table 5. But their gaps are not so large. Moreover, although the CPU times spent by the three approaches generally increase with the instance size N , the CPU times spent by CPLEX generally have a very sharper growth than those spent by our algorithm and Yan's algorithm, especially for large-size instances. From these results, we can see that our proposed algorithm has a better computational performance than CPLEX.

Furthermore, Table 6 reports the comparison results of our proposed algorithm with and without a repairing procedure. From it, we can see that our algorithm without a repairing procedure (denoted by Our[#]) generally performs worse on instances Group 1 and Group 2 than that with a repairing procedure (denoted by Our), in terms of solution quality. From column "AD," we can see that when problem size (i.e., N) increases, our algorithm has a much better solution than that without a repairing procedure, although it costs relatively much more CPU time. In other words, our designed repairing procedure for infeasible solutions is

effective for the studied problem. Based on the above results and analysis, we can conclude that our proposed algorithm works well and is effective to solve the studied scheduling problem.

5. Conclusions

This paper proposed a hybrid QEA with improved decoding scheme to solve a single robot one-degree cyclic scheduling problem with flexible processing times in a robotic flow shop. In particular, three different decoding procedures were proposed to convert Q-bit individual into robot move sequences. A repairing procedure was designed to repair the infeasible sequence. Both Q-gate and adaptive genetic operators as variant operators were applied to evolve the population. The effectiveness of the proposed algorithm was demonstrated by solving benchmark instances and randomly generated instances compared with commercial software CPLEX and Yan's algorithm. Experimental results indicate that our decoding scheme outperforms the shifting decoding scheme, and the proposed algorithm can provide high-quality solutions within a reasonable time. The results also imply that the proposed algorithm generally has a shorter computation time than CPLEX, especially for large-size instances and has a better solution quality than Yan's algorithm.

For further research direction, the hybrid QEA algorithm developed in this study can be extended to the robotic scheduling problem with multiobjective optimization (such as simultaneously optimizing of cycle time and production cost). This extension can be accomplished by applying the well-known Pareto-based ranking approach to evaluate each individual. In further research, we may extend the proposed algorithm to solve the robotic scheduling problem in a multidegree context.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was partially supported by the scientific research program funded by Xi'an University of Science and Technology (2015QDJ052, 201749), the National Natural Science Foundation of China (71401133, 71601154), and China Scholarship Council.

References

- [1] Q. Zhang, H. Manier, M.-A. Manier, and C. Bloch, "Heuristics for predictive hoist scheduling problems," *European Journal of Industrial Engineering*, vol. 8, no. 5, pp. 695–715, 2014.
- [2] H. Chen, C. Chu, and J.-M. Proth, "Cyclic scheduling of a hoist with time window constraints," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 1, pp. 144–152, 1998.
- [3] M.-A. Manier and C. Bloch, "A classification for hoist scheduling problems," *International Journal of Flexible Manufacturing Systems*, vol. 15, no. 1, pp. 37–55, 2003.
- [4] P. Yan, A. Che, X. Cai, and X. Tang, "Two-phase branch and bound algorithm for robotic cells rescheduling considering limited disturbance," *Computers and Operations Research*, vol. 50, pp. 128–140, 2014.
- [5] J. Feng, A. Che, and N. Wang, "Bi-objective cyclic scheduling in a robotic cell with processing time windows and non-Euclidean travel times," *International Journal of Production Research*, vol. 52, no. 9, pp. 2505–2518, 2014.
- [6] L. Lei and T. Wang, "A proof: the cyclic hoist scheduling problem is NP-complete," Working Paper, Rutgers University, 1989.
- [7] L. W. Phillips and P. S. Unger, "Mathematical programming solution of a hoist scheduling program," *AIIE Transactions*, vol. 8, no. 2, pp. 219–225, 1976.
- [8] A. Che, J. Feng, H. Chen, and C. Chu, "Robust optimization for the cyclic hoist scheduling problem," *European Journal of Operational Research*, vol. 240, no. 3, pp. 627–636, 2015.
- [9] L. Lei and T. J. Wang, "Determining optimal cyclic hoist schedules in a single-hoist electroplating line," *IIE Transactions*, vol. 26, no. 2, pp. 25–33, 1994.
- [10] P. Yan, A. Che, N. Yang, and C. Chu, "A tabu search algorithm with solution space partition and repairing procedure for cyclic robotic cell scheduling problem," *International Journal of Production Research*, vol. 50, no. 22, pp. 6403–6418, 2012.
- [11] J. Liu, Y. Jiang, and Z. Zhou, "Cyclic scheduling of a single hoist in extended electroplating lines: a comprehensive integer programming solution," *IIE Transactions*, vol. 34, no. 10, pp. 905–914, 2002.
- [12] A. Che and C. Chu, "Cyclic hoist scheduling in large real-life electroplating lines," *OR Spectrum*, vol. 29, no. 3, pp. 445–470, 2007.
- [13] L. Lei and T. Wang, "The minimum common-cycle algorithm for cyclic scheduling of two material handling hoists with time window constraints," *Management Science*, vol. 37, no. 12, pp. 1629–1638, 1991.
- [14] Z. Zhou and J. Liu, "A heuristic algorithm for the two-hoist cyclic scheduling problem with overlapping hoist coverage ranges," *IIE Transactions*, vol. 40, no. 8, pp. 782–794, 2008.
- [15] M.-A. Manier, C. Varnier, and P. Baptiste, "Constraint-based model for the cyclic multi-hoists scheduling problem," *Production Planning and Control*, vol. 11, no. 3, pp. 244–257, 2000.
- [16] M.-A. Manier and S. Lamrous, "An evolutionary approach for the design and scheduling of electroplating facilities," *Journal of Mathematical Modelling and Algorithms*, vol. 7, no. 2, pp. 197–215, 2008.
- [17] A. Che and C. Chu, "Single-track multi-hoist scheduling problem: a collision-free resolution based on a branch-and-bound approach," *International Journal of Production Research*, vol. 42, no. 12, pp. 2435–2456, 2004.
- [18] J. M. Y. Leung, G. Zhang, X. Yang, R. Mak, and K. Lam, "Optimal cyclic multi-hoist scheduling: a mixed integer programming approach," *Operations Research*, vol. 52, no. 6, pp. 965–976, 2004.
- [19] A. Elmi and S. Topaloglu, "Multi-degree cyclic flow shop robotic cell scheduling problem with multiple robots," *International Journal of Computer Integrated Manufacturing*, pp. 1–17, 2016.
- [20] A. El Amraoui, M.-A. Manier, A. El Moudni, and M. Benrejeb, "A mixed linear program for a multi-part cyclic hoist scheduling problem," *International Journal of Science and Techniques of Automatic Control & Computer Engineering*, vol. 11, pp. 612–623, 2008.

- [21] J. Feng, A. Che, and C. Chu, "Dynamic hoist scheduling problem with multi-capacity reentrant machines: a mixed integer programming approach," *Computers and Industrial Engineering*, vol. 87, article 4074, pp. 611–620, 2015.
- [22] A. Che, Z. Zhou, C. Chu, and H. Chen, "Multi-degree cyclic hoist scheduling with time window constraints," *International Journal of Production Research*, vol. 49, no. 19, pp. 5679–5693, 2011.
- [23] A. Elmi and S. Topaloglu, "Multi-degree cyclic flow shop robotic cell scheduling problem: ant colony optimization," *Computers and Operations Research*, vol. 73, pp. 67–83, 2016.
- [24] X. Li and R. Y. K. Fung, "A Mixed integer linear programming solution for single hoist Multi-degree cyclic scheduling with reentrance," *Engineering Optimization*, vol. 46, no. 5, pp. 704–723, 2014.
- [25] D. Deutsch, "Quantum theory, the Church-Turing principle and the universal quantum computer," *Proceedings of the Royal Society. London. Series A*, vol. 400, no. 1818, pp. 97–117, 1985.
- [26] A. Narayanan and M. Moore, "Quantum-inspired genetic algorithms," in *Proceedings of the IEEE International Conference on Evolutionary Computation*, p. 6166, Nagoya, Japan, May 1996.
- [27] K.-H. Han and J.-H. Kim, "Quantum-inspired evolutionary algorithm for a class of combinatorial optimization," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 6, pp. 580–593, 2002.
- [28] T.-C. Lu and G.-R. Yu, "An adaptive population multi-objective quantum-inspired evolutionary algorithm for multi-objective 0/1 knapsack problems," *Information Sciences*, vol. 243, pp. 39–56, 2013.
- [29] A. Che, P. Wu, F. Chu, and M. Zhou, "Improved quantum-inspired evolutionary algorithm for large-size lane reservation," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 45, no. 12, pp. 1535–1548, 2015.
- [30] B.-B. Li and L. Wang, "A hybrid quantum-inspired genetic algorithm for multiobjective flow shop scheduling," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 37, no. 3, pp. 576–591, 2007.
- [31] T. Zheng and M. Yamashiro, "Solving flow shop scheduling problems by quantum differential evolutionary algorithm," *International Journal of Advanced Manufacturing Technology*, vol. 49, no. 5, pp. 643–662, 2010.
- [32] J. Gu, X. Gu, and M. Gu, "A novel parallel quantum genetic algorithm for stochastic job shop scheduling," *Journal of Mathematical Analysis and Applications*, vol. 355, no. 1, pp. 63–81, 2009.
- [33] J. Gu, M. Gu, C. Cao, and X. Gu, "A novel competitive co-evolutionary quantum genetic algorithm for stochastic job shop scheduling problem," *Computers and Operations Research*, vol. 37, no. 5, pp. 927–937, 2010.
- [34] K.-H. Han and J.-H. Kim, "Quantum-inspired evolutionary algorithms with a new termination criterion, H_ϵ gate, and two-phase scheme," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 2, pp. 156–169, 2004.
- [35] S. Akpınar and G. Mirac Bayhan, "A hybrid genetic algorithm for mixed model assembly line balancing problem with parallel workstations and zoning constraints," *Engineering Applications of Artificial Intelligence*, vol. 24, no. 3, pp. 449–457, 2011.
- [36] M. Srinivas and L. M. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24, no. 4, pp. 656–667, 1994.
- [37] P. Yan, G. Wang, A. Che, and Y. Li, "Hybrid discrete differential evolution algorithm for biobjective cyclic hoist scheduling with reentrance," *Computers and Operations Research*, vol. 76, pp. 155–166, 2016.
- [38] M.-A. Manier, *Contribution à l'ordonnancement cyclique du système de manutention d'une ligne de galvanoplastie [PhD thesis in Automatics and Computer Science]*, Université de Franche-Comté, Besançon, France, 1994.
- [39] Z. Zhou, A. Che, and P. Yan, "A mixed integer programming approach for multi-cyclic robotic flowshop scheduling with time window constraints," *Applied Mathematical Modelling*, vol. 36, no. 8, pp. 3621–3629, 2012.