Expert Systems With Applications 187 (2022) 115978

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Semiconductor final testing scheduling using Q-learning based hyper-heuristic

Jian Lin, Yang-Yuan Li, Hong-Bo Song [*]

School of Information Management and Artificial Intelligence, Zhejiang University of Finance and Economics, Hangzhou 310018, Zhejiang, China

ARTICLE INFO

Keywords:
Q-learning
Hyper-heuristic
Semiconductor final testing
Scheduling

ABSTRACT

Semiconductor final testing scheduling problem (SFTSP) has extensively been studied in advanced manufacturing and intelligent scheduling fields. This paper presents a Q-learning based hyper-heuristic (QHH) algorithm to address the SFTSP with makespan criterion. The structure of QHH employs the Q-learning algorithm as the high-level strategy to autonomously select a heuristic from a pre-designed low-level heuristic set. The selected heuristic in different stages of the optimization process is recognized as the executable action and performed on the solution space for better results. An efficient encoding and decoding pair is presented to generate feasible schedules, and a left-shift scheme is embedded into the decoding process for improving resources utilization. Additionally, the design-of-experiment method is implemented to investigate the effect of parameters setting. Both computational simulation and comparison are finally carried out on a benchmark set and the results demonstrate the effectiveness and efficiency of the proposed QHH.

1. Introduction

In the past decades, manufacturing systems have received great attention due to its practical applications (Brucker & Schlie, 1990; Ruiz & Stützle, 2007; Remigio & Swartz, 2020). As an important high-tech industry, semiconductor manufacturing is facing fierce competition with the rapid growth of chip demands. Typical examples include mobile devices, LED, green energy, car electronics, etc (Hao, Wu, Chien, & Gen, 2014). In a semiconductor factory, a fully functional integrated circuit (IC) experiences four fabrication processes including wafer fabrication, wafer probe, packaging and final testing, and each of them is confronted with different challenges (Uzsoy, Church, Ovacik, & Hinchman, 1992). Particularly, the final testing is deemed as the bottleneck process and has been the most studied topic because of its complexity, uncertainty and time-consuming (Chiang, Guo, & Pai, 2008). The final testing process is to examine malfunctions and guarantee the quality of ICs before delivering it to customers (Joung, He, Yoon, Vancheeswaran, Abela, & Andres, 2017). However, since the testing resources and machines are expensive and product-specific, the budget and available quantities of them are quite limited (Wu, Hao, Chien, & Gen, 2012). It is vital for companies to determine a satisfactory testing sequence for all the ICs to meet the requirements and qualifications within a promised due date while utilizing resources efficiently (Gao, Bard, Chacon, & Stuber, 2015). Hence, the semiconductor final testing scheduling problem (SFTSP) is an important topic that deserves investigation.

Some research results reveal that the SFTSP involves distinctive features of the regular job shop problems (JSPs), and determining a best schedule for this problem is NP-hard (Wu & Chien, 2008). Obviously, deterministic methods are not suitable in solving this intractable problem (He, Joung, Yoon, Vancheeswaran, & Andres, 2016). After the first effort made by Uzsoy, Martin-Vega, Lee, and Leonard (1991), many approaches have been proposed to solve the SFTSP with various optimization models, and heuristics and meta-heuristics are commonly-used methods. Uzsoy, Lee, and Martin-Vega (1992) presented several heuristics to minimize the maximum lateness as well as the number of tardy jobs for a single machine SFTSP. Later, Uzsoy, Church, Ovacik, and Hinchman (1993) evaluated the performance of several dispatching rules for the SFTSP and examined the effects of uncertainties in problem data and job arrival patterns. After that, Ovacik and Uzsoy (1996) exploited the structure of the routings in semiconductor testing process to present a decomposition method which uses specialized procedures to schedule a number of work centers. Freed and Leachman (1999) presented an enumeration solution program for a multi-head tester scheduling problem, and they also illustrated the problems of previously suggested tester scheduling algorithms. Pearn, Chung, Chen, and Yang (2004) proposed three fast network algorithms to minimize the total workload for a SFTSP with batch processing stage, job clusters and other

[*] Corresponding author.
E-mail address: di7ganshb@163.com (H.-B. Song).

https://doi.org/10.1016/j.eswa.2021.115978
Received 3 February 2021; Received in revised form 26 July 2021; Accepted 23 September 2021
Available online 28 September 2021
0957-4174/© 2021 Elsevier Ltd. All rights reserved.

Nomenclature

n	number of jobs
m	number of machines
P_i	number of operations of J_i
η_1	number of tester types
η_2	number of handler types
η_3	number of accessory types
H	number of low-level heuristics
λ	total number of all resource types, $\lambda = \eta_1 + \eta_2 + \eta_3$
t	index of time, $t \geqslant 0$
J_i	index of jobs, $i = 1, 2, \cdots, n$
M_j	index of machines, $j = 1, 2, \cdots, m$
$O_{i,p}$	index of operations, $p = 1, 2, \cdots, P_i$
$S_{i,p}$	the starting time of $O_{i,p}$
$F_{i,p}$	the finishing time of $O_{i,p}$
$ST_{j,l}$	the setup time from machine M_j to M_l
$D_{i,p,j}$	the processing time of operation $O_{i,p}$ assigned to M_j
ζ	the λ-dimensional vector indicating the quantity of each type of resources
Γ_j	the λ-dimensional binary vector determining whether the uth type resource is utilized by M_j, $u = 1, 2, \cdots, \lambda$
$x_{i,p,j}$	the binary decision variable determining whether $O_{i,p}$ is processed on M_j
$y_{i,j,t}$	the binary decision variable determining whether J_i is processed on M_j at time t
$z_{j,t}$	the binary decision variable determining whether M_j is utilized at time t

Abbreviations

RL	reinforcement learning
LLH	low-level heuristic
SFTSP	semiconductor final testing scheduling problem
CS	cuckoo search algorithm
CSRS	cuckoo search algorithm with reinforcement learning and surrogate modeling
nFOA	novel fruit fly optimization algorithm
HEDA	hybrid estimation of distribution algorithm

complex constraints. Lin, Wang, and Lee (2004) developed a capacity-constrained scheduling algorithm to maximize the committed volume performance for semiconductors logic IC final test operations.

Recently, meta-heuristic algorithms have been applied to address the SFTSP. Wu and Chien (2008) modeled the SFTSP as a JSP with limited simultaneous multiple resources, and presented a genetic algorithm (GA) to minimize the makespan. Further, Wu et al. (2012) developed a bi-vector genetic algorithm (bvGA) which uses a two-list based encoding scheme to represent operation sequence and resource assignment. By incorporating the co-evolutionary framework, Hao et al. (2014) proposed a cooperative estimation of distribution algorithm (CEDA) with the advantage of maintaining population diversity during the evolutionary process. These algorithms achieve significantly better performance than the heuristic methods with the sacrifice of computational efficiency. Thus, several researches were motivated for SFTSP from the aspect of balancing the performance and computational efficiency. A fruit fly optimization algorithm (nFOA) and a hybrid estimation of distribution algorithm (HEDA) were presented by Zheng, Wang, and Wang (2014) and Wang, Wang, Liu, and Xu (2015), respectively. A knowledge-based multi-agent evolutionary algorithm (KMEA) was proposed by Wang and Wang (2015). It is worth pointing out that the computational efficiency for these algorithms are greatly improved due to the reduced complexity and less control parameters. More recently, a cooperative co-evolutionary invasive weed optimization (CCIWO) algorithm was proposed by Sang, Duan, and Li (2018), in which the sizes of colonies in CCIWO are set to be independent to balance the search abilities. However, the mentioned meta-heuristic algorithms for SFTSP are parameter-sensitive and it is a tedious process to select proper parameters for them. Thus, Cao, Lin, Zhou, and Huang (2019) presented a cuckoo search algorithm with reinforcement learning (RL) to simplify the process of parameters selection. Nevertheless, meta-heuristics trend to be problem-specific or not suitable for all instances in the same problem based on the No-Free-Lunch theorem (Wolpert & Macready, 1997).

Hyper-heuristic is regarded as a possible way to develop methods that are more generally applicable than other implementations of search methodologies (Branke, Nguyen, Pickardt, & Zhang, 2016). The main distinction between meta-heuristic and hyper-heuristic is that the former operates directly on the solution domain, while the latter manipulates a set of pre-designed low-level heuristics (LLHs) (Rajni & Chana, 2013; Lin, 2019). Hyper-heuristics can be categorized into two classes: heuristic generation that produces new heuristics from components of existing LLHs, and heuristic selection that comprises heuristics selected from the existing ones (Kheiri & Keedwell, 2017; Lin, 2019; Drake, Kheiri, Özcan, & Burke, 2020). Some current state-of-the-art methods such as machine learning (Asta, Özcan, & Curtois, 2016), RL (Wauters, Verbeeck, Causmaecker, & Berghe, 2013), harmony search (Dempster & Drake, 2016), variable neighborhood search (Hsiao, Chiang, & Fu, 2011), greedy gradient heuristic (Kalender, Kheiri, Özcan, & Burke, 2013) and multi-armed bandit selection (Sabar, Ayob, Kendall, & Rong Qu, 2015) have been employed as the selection strategy in hyper-heuristic to solve various problems. Among them, the RL technique, which does not require the time-consuming meta-optimization process, has its unique advantages to intelligently guide the selection of LLHs (Adriaensen, Brys, & Nowé, 2014; Choong, Wong, & Lim, 2018). Besides, the RL based hyper-heuristic has demonstrated its excellent performance on various scheduling problems (Choong et al., 2018).

The aforementioned insights inspire the research of designing a Q-learning based hyper-heuristic (QHH) scheme to solve the SFTSP, which has not been investigated to the best of the authors' knowledge. QHH belongs to selection hyper-heuristic where an initial solution is iteratively improved during the optimization process. As one of the most widely used strategies in RL, Q-learning technique is employed as the selection method and operates on heuristic space to guide the selection of LLHs during different stages of the optimization process. The selected LLH is served as the action to perform on the solution domain to find better solutions. After the execution period, the LLH receives a reward or penalty based on a reward function. By using the Q-function, a Q-value can be obtained to evaluate this execution and stored in a Q-table. According to Q-table, the solution learns which action will be chosen in the next stage. The parameters involved in QHH are calibrated through experiments designed by the design-of-experiment (DOE) method (Montgomery, 2008). The effectiveness of QHH is verified via a set of benchmark instances from practical semiconductor final testing process.

The main contributions of this paper can be described as follows: (1) An efficient encoding and decoding pair is presented to generate feasible schedules. (2) A left-shift method is incorporated into the proposed decoding scheme to further minimize the makespan. (3) Eight simple heuristics are utilized to construct the LLH set and defined as the executable actions. (4) The Q-learning technique is employed to guide the selection of LLHs during different stages of the optimization process. (5) The DOE method is implemented to suggest appropriate values of parameters for experiments. Computational results and comparisons demonstrate the superiority of the proposed algorithm.

The rest of the paper is organized as follows: The description of

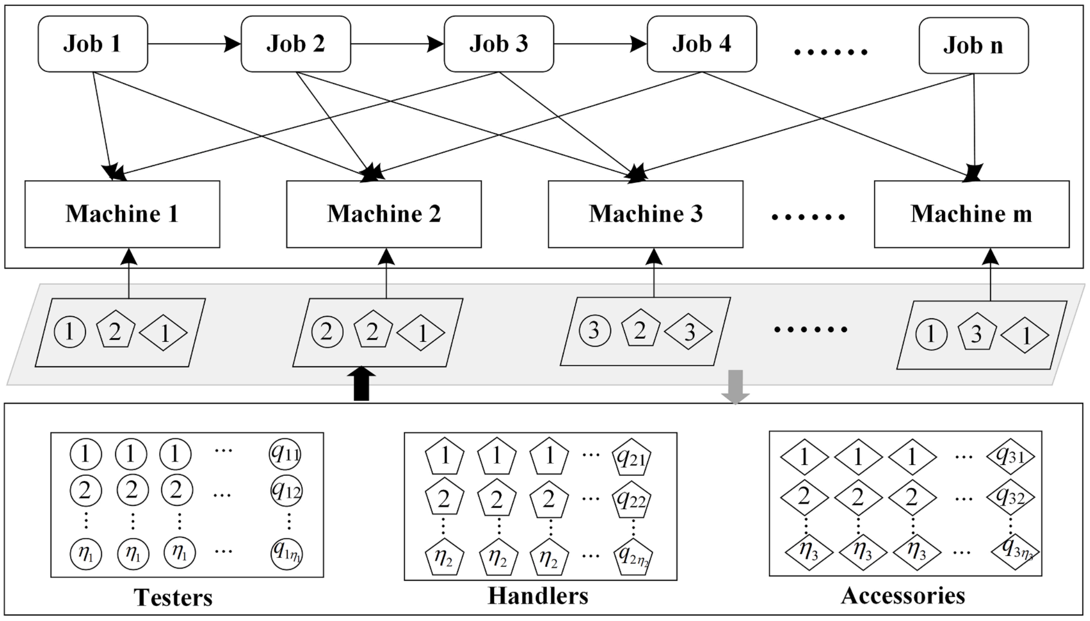

Fig. 1. Illustration of the SFTSP.

SFTSP and classical Q-learning algorithm are introduced in Sections 2 and 3, respectively. Section 4 designs the QHH structure in detail. Computational results on testing instances together with comparison to some state-of-the-art algorithms are analyzed in Section 5, and the conclusion is drawn in Section 6.

2. Problem description

The SFTSP considered in this paper can be viewed as a flexible job-shop scheduling problem with multi-resource constraint and setup time. In this section, the SFTSP will be presented first and a mathematical MILP model is then formulated. The notations used in the optimization model are given in the nomenclature.

SFTSP can be illustrated as in Fig. 1, where n independent jobs $J = \{J_1, J_2, \cdots J_n\}$ are tested on m machines $M = \{M_1, M_2, \cdots M_m\}$. A job J_i contains a set of sequential operations $\{O_{i,1}, O_{i,2}, \cdots, O_{i,P_i}\}$ and P_i is total number of operations for J_i. Each operation can be executed on one or more machines and the processing times for them are predetermined and assumed to be integers. The resources used in the testing process include η_1 types of testers, η_2 types of handlers and η_3 types of accessories. The available quantities of testers, handlers and accessories with different types are $\{q_{11}, q_{12}, \cdots q_{1\eta_1}\}$, $\{q_{21}, q_{22}, \cdots q_{2\eta_2}\}$ and $\{q_{31}, q_{32}, \cdots q_{3\eta_3}\}$, respectively. A machine is associated with a specific resource combination of certain types of tester, handler and accessory, and can only be activated when the required resources are available. The resource type combination for each machine is known in advance and can be represented as a vector with three elements. As shown in Fig. 1, the resources required for machine M_1 are tester with type 1, handler with type 2 and accessory with type 1, respectively, and the corresponding resource type combination vector is [1, 2, 1]. Note that the amounts of resources are always limited because of their high expenses and this is also one of the key constraints in SFTSP.

In addition, setup time is always required when changing an operation from one machine to another. This means that the setup time for SFTSP is unrelated to operation, and only depends on the current machine and the next selected machine. Thus a changeover matrix with dimension $m \times m$ is used to represent the setup times for all machines. The testing process can be started if and only if a job arrives, all the needed resources are available, setup process is completed and operation precedence constraints are satisfied.

The following assumptions are also made for SFTSP: (a) Preemption is not allowed once an operation starts. (b) There are no precedence constraints among different jobs. (c) Each machine can only process one job at a time. (d) Each operation in a job must be processed once. (e) Jobs, machines and resources are available initially. Based on the above description, the MILP model for SFTSP is established as follows:

$$\text{Minimize } C_{max} = \max F_{i,p},\ p = 1, 2, 3 \cdots P_i \tag{1}$$

Subject to:

$$S_{i,p} \geqslant F_{i,p-1},\ i \in J,\ p = 2, \cdots P_i \tag{2}$$

$$S_{i,p} \geqslant F_{i,p-1} + ST_{j,l},\ i \in J,\ j, l \in M \wedge j \neq l,\ p = 2, 3, 4, \cdots P_i \tag{3}$$

$$\sum_{j}^{m} z_{j,t} \times \Gamma_j \leqslant \zeta, j \in M, t \geqslant 0 \tag{4}$$

$$\sum_{i}^{n} y_{i,j,t} \leqslant 1,\ i \in J,\ j \in M, t \geqslant 0 \tag{5}$$

$$\sum_{j}^{m} x_{i,p,j} = 1,\ i \in J,\ j \in M,\ p = 1, 2, 3, \cdots P_i \tag{6}$$

$$x_{i,p,j} = \{0, 1\},\ y_{i,j,t} = \{0, 1\},\ z_{j,t} = \{0, 1\},\ j \in M,\ i \in J,\ t \geqslant 0 \tag{7}$$

$$S_{i,p} \geqslant 0,\ t \geqslant 0,\ i \in J,\ p = 1, 2, 3, \ldots P_i \tag{8}$$

The optimization objective is given in Eq. (1) and the constraints are listed as in Eqs. (2)–(8). Eq. (1) guarantees that the makespan C_{max} depends on the latest finishing time of all operations. Eq. (2) restricts that the precedence constraints should be satisfied by all the operations in a job. Eq. (3) demands that the setup time $ST_{j,l}$ is needed if M_j and M_l are selected to execute $O_{i,p-1}$ and $O_{i,p}$, respectively. The resource constraint is presented in Eq. (4) and it should be satisfied at any time slot. Note that Γ_j is a λ-dimensional binary vector and the uth element of Γ_j is set to 1 if the uth type resource has been utilized. ζ is a λ-dimensional vector indicating the available quantities of different resource types, and $\zeta = [q_{11}, \cdots q_{1\eta_1}, q_{21}, \cdots q_{2\eta_2}, q_{31}, \cdots q_{3\eta_3}]$. Eqs. (5) and (6) ensure that each machine can process at most one job at a time and each job should be processed once, respectively. The feasible spaces of decision variables are determined by Eqs. (7) and (8), where $x_{i,p,j}$, $y_{i,j,t}$ and $z_{j,t}$ are binary

Table 1
Processing time table.

Job ID	Operation ID	Machine ID		
		M_1	M_2	M_3
J_1	$O_{1,1}$	2	3	–
	$O_{1,2}$	3	–	–
J_2	$O_{2,1}$	4	5	3
	$O_{2,2}$	6	5	–
J_3	$O_{3,1}$	–	5	7
	$O_{3,2}$	–	6	4

Table 2
Setup time table.

Machine ID	M_1	M_2	M_3
M_1	0	1	2
M_2	1	0	2
M_3	3	1	0

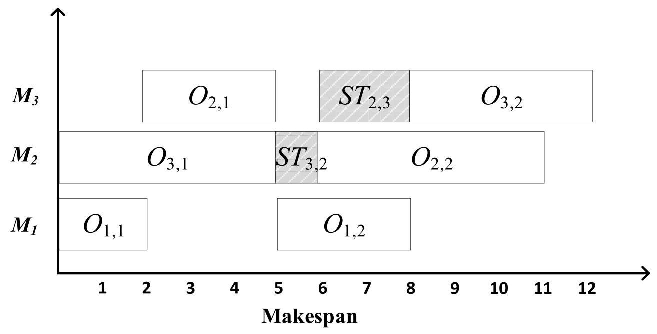

Fig. 2. Gantt chart for a feasible solution to the example.

decision variables, and t and $S_{i,p}$ should be positive.

An example with 3 jobs and 3 machines is provided to explain the SFTSP more clearly. The processing time of each operation and the setup times between different machines are listed in Tables 1 and 2, respectively. Assume the testers, handlers and accessories all have two types, and the available quantities of them are {1, 1}, {2, 1} and {2, 2}, respectively. The resource combinations for machines M_1, M_2 and M_3 are [1, 1, 1], [2, 2, 1] and [1, 1, 2], respectively. The operation sequencing vector of the instance is $\{O_{1,1}, O_{3,1}, O_{2,1}, O_{1,2}, O_{2,2}, O_{3,2}\}$, and the corresponding machine assignment vector is assumed to be $\{M_1, M_2, M_3, M_1, M_2, M_3\}$.

The Gantt chart for a feasible solution of the instance is illustrated in Fig. 2. As seen in the figure, $O_{1,1}$ and $O_{3,1}$ are processed on M_1 and M_2 at the starting time because the required resources are available. However, $O_{2,1}$ cannot be processed at the starting time on M_3 since that the quantity of tester with type 1 is one and has already been used by M_1. Hence, the starting time of $O_{2,1}$ has to delay until the required resource is released, that is, $O_{1,1}$ has been finished. Similarly, we can see that $O_{3,2}$ starts processing when $O_{1,2}$ is finished due to the resource constraint. On the other hand, the operations of $O_{2,1}$ and $O_{2,2}$ are processed on M_3 and M_2, respectively, and setup time $ST_{3,2}$ is needed between the two operations. The same situation occurred on the operations of $O_{3,1}$ and $O_{3,2}$. Finally, the makespan of this schedule can be obtained as 12.

3. Q-learning algorithm

As one of the most efficient techniques in RL methods, Q-learning aims to acquire an optimal behavior for achieving the ideal state by the interactions of reward and penalty between an agent and a dynamic environment (Watkins & Dayan, 1992; Shen, Minku, Marturi, Guo, & Han, 2018). In Q-learning, each state-action pair is given a total cumulative reward value (denoted as a Q-value), which is measured by a Q-function as in Eq. (9). Suppose that $S = \{s|s_1, s_2, \cdots, s_n\}$ denotes a set of possible states, $A = \{a|a_1, a_2, \cdots, a_m\}$ denotes a set of selectable actions, r_t denotes the immediate reinforcement signal, $\alpha \in [0, 1]$ denotes the learning rate, $\gamma \in [0, 1]$ denotes the discount factor, and $Q_t(s_t, \quad a_t)$ denotes the Q-value at time t. The Q-values of all state-action pairs are stored in a Q-table which is employed as a reference to learn the expected reward value when facing different situations in future. Q-learning algorithm is extensively adopted in various scenarios, such as the motions of a mobile robot (Jaradat, Al-Rousan, & Quadan, 2011), the optimal battery management (Wei, Liu, & Shi, 2015), job shop scheduling (Shu, 2020) and the tracking problem of unknown discrete-time linear systems (Kiumarsi, Lewis, Modares, Karimpour, & Naghibi-Sistani, 2014), etc.

$$Q_{t+1}(s_t, \quad a_t) = Q_t(s_t, \quad a_t) + \alpha_t\{r_t + \gamma \underset{a \in A}{\operatorname{argmax}} \quad Q(s_{t+1}, \quad a) - Q_t(s_t, \quad a_t)\} \tag{9}$$

Combining the above descriptions, the Q-learning procedure can be summarized as follows:

Step 1: Initialize the Q-table for each state-action pair;
Step 2: Identify the current state s_t;
Step 3: Select an action a_t using the action selection strategy;
Step 4: Execute the action a_t, receive the signal r_t and determine the next state s_{t+1};
Step 5: Calculate $Q_{t+1}(s_t, \quad a_t)$ using the Q-function as given in Eq. (9);
Step 6: Go back to Step 2 until the termination condition is satisfied.

4. Proposed QHH algorithm

In this section, a Q-learning based hyper-heuristic (QHH) algorithm is proposed for solving the SFTSP. First, the encoding and decoding schemes are proposed to represent solution and obtain feasible schedules, respectively. Then, some problem-specific heuristics are designed to construct a set of low-level heuristics (LLHs) which are also deemed as the executable actions in QHH scheme. In addition, Q-learning is employed as the high-level strategy to select appropriate action during the optimization process, and the state representation and action selection method are also introduced. Finally, the algorithmic framework of QHH is presented in the last subsection.

4.1. Solution encoding and decoding schemes

The bi-vector-based encoding scheme is frequently used to represent

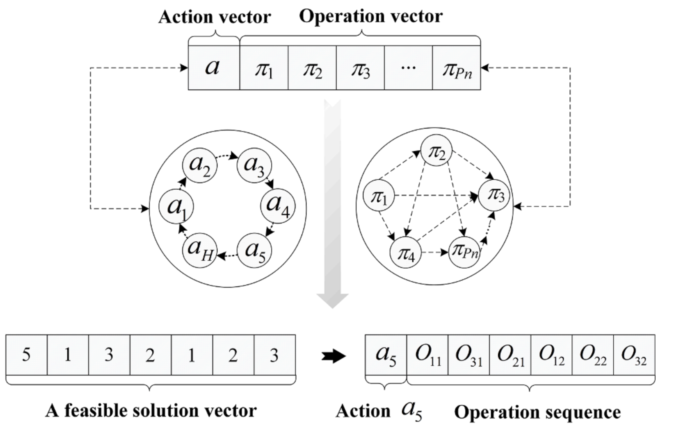

Fig. 3. Illustration of solution encoding scheme.

```
Procedure Decoding scheme
    Input: an operation vector $\pi$;
    Output: a feasible schedule $\Omega$;
    Initialize:
        $C_\pi = 0$;
        For $j = 1$ to $m$
            $C_j = 0$, $w_j = 0$, $\Omega_j = \phi$;
        Next For
    End of initialization
    For $i = 1$ to $n$
        $flag = false$, $C_M = \infty$;
        For $j = 1$ to $m$
            Find the machine $M_j$ that can process operation $\pi_i$;
            Identified the setup time $ST_{j',j}$ on $M_j$;
            If an idle time interval $[t_j^S, t_j^E]$ is found on $M_j$
                Perform the left-shift method and set $flag = True$;
                $C_{i,p,j} = S_{i,p} + ST_{j',j} + D_{i,p,j}$;
                $C_{temp} = C_j$;
            End If
            If $flag$ is $false$
                $C_{i,p,j} = C_j + ST_{j',j} + D_{i,p,j}$;
                $C_{temp} = C_{i,p,j}$;
            End If
            If $C_{temp} < C_M$
                $C_M = C_{i,p,j}$;
                $j" = j$;
            End If
        Next For
        $C_{j"} = C_{temp}$, $w_{j"} = w_{j"} + 1$, $\Omega_{j"}(w_{j"}) = \pi_i$;
        If $C_M > C_\pi$
            $C_\pi = C_M$;
    Next For
End procedure
```

Fig. 4. Pseudo code of the proposed decoding scheme.

solutions for SFTSP (Wang et al., 2015; Cao et al., 2019), where operation sequence and machine assignment are simultaneously considered. In QHH, an operation vector $\pi = \{\pi_1, \pi_2, \pi_3 \dots \pi_{P_n}\}$ and an action vector a are encoded into a single solution vector. Especially, the corresponding job number is used to denote the operation of a job, and thus the kth occurrence of a job number in operation vector refers to the kth operation of the job. A feasible solution of the example in Table 1 is illustrated in Fig. 3, where the first number of the vector indicates that the action a_5 is selected to perform on solution domain, and the remaining part $\{1, 3, 2, 1, 2, 3\}$ is the operation vector that determines the processing order for all operations. The makespan value obtained by the decoding scheme is used as the fitness to evaluate the solution.

Solution decoding means to generate a feasible schedule by allocating operations to appropriate machines while satisfying precedence and resource constraints. However, some idle time intervals will be inevitably appeared between two adjacent operations on a machine. In order to improve machine utilization and further minimize the makespan, an effective local search strategy called left-shift method (Wang & Zheng, 2018; Lin, Zhu, & Gao, 2020) is embedded into the decoding process. The main idea of the method is that operations should be processed on machines as compact as possible to reduce the idle time. Assume that operation $O_{i,p}$ is available to be assigned to machine M_j, then we need to find an appropriate idle time interval $[t_j^S, t_j^E]$ on M_j so as to process $O_{i,p}$. The operation $O_{i,p}$ can be processed on M_j between t_j^S and t_j^E when the following constraints are satisfied:

$$t_j^E \geqslant S_{i,p} + D_{i,p,j}, j \in M, i \in J, p = 1, 2 \cdots P_i \quad (10)$$

$$S_{i,p} = \max\left\{t_j^S, S_{i,p-1} + D_{i,p-1,j'}\right\}, j' \in M, i \in J, p = 2, 3 \cdots P_i \quad (11)$$

where t_j^E and t_j^S are the finishing time and starting time of idle time intervals, respectively. $D_{i,p,j}$ is the processing time of operation $O_{i,p}$ processed on M_j. $S_{i,p}$ represents the earliest starting time of $O_{i,p}$ and can be calculated as in Eq. (11). Note that operation $O_{i,p}$ can be started only if all of its predecessors are finished.

By using the left-shift method, an efficient decoding scheme is proposed to generate schedules efficiently. For a solution, the decoding process is performed by assigning operations to the machines with the minimum makespan. If the required resources are insufficient at a moment, the operation should wait until all the resources are available. The illustrative procedure is described in Fig. 4, where w_j represents the number of operations on machine M_j, Ω_j is a set of operations processed on machine M_j, ϕ denotes an empty set, $flag$ is a Boolean variable determining whether the left-shift method is performed, C_{temp} is a temporary variable, $C_{i,p,j}$ is the completion time of $O_{i,p}$ on M_j, C_M is the minimum completion time of $O_{i,p}$ on all available machines, C_j is the completion time of the final operation on M_j, and C_π is the makespan of π.

4.2. Action representation

The design of LLHs dramatically affects the efficiency of the hyper-heuristic algorithm. In this section, eight simple and easy-to-implement heuristics (Lin, Wang, & Li, 2017; Lin et al., 2020) are designed and regarded as different actions.

(1) ***Two-point-swap*** (***LLH$_1$***): Select two different operations randomly from the sequence and swap them.
(2) ***Forward-insert*** (***LLH$_2$***): Select two different operations $O_{i,p}$ and $O_{j,p}$ $(j > i)$ randomly from the sequence and insert $O_{j,p}$ before $O_{i,p}$.
(3) ***Backward-insert*** (***LLH$_3$***): Select two different operations $O_{i,p}$ and $O_{j,p}$ $(j > i)$ randomly from the sequence and insert $O_{j,p}$ behind $O_{i,p}$.
(4) ***Adjacent-swap*** (***LLH$_4$***): Select a position i randomly from the sequence, and swap it with the operation on position $i + 1$.
(5) ***Two-binding-exchange*** (***LLH$_5$***): Select an operation $O_{i,p}$ randomly from the sequence, then exchange $O_{i,p}$ with $O_{i+3,p}$ and $O_{i+1,p}$ with $O_{i+2,p}$.
(6) ***Half-side-exchange*** (***LLH$_6$***): Divide the sequence into two equal parts and select two different operations $O_{i,p}$ and $O_{j,p}$ $(j \neq i)$ from the left half-side, while $O_{k,p}$ and $O_{l,p}$ $(k \neq l)$ from another side. Then, swap $O_{i,p}$ with $O_{k,p}$ and $O_{j,p}$ with $O_{l,p}$.
(7) ***Subsequence-inverse*** (***LLH$_7$***): Select two different positions i and j $(j > i)$ randomly from the sequence, then inverse the subsequence $\{O_{i,p}, O_{i+1,p} \cdots O_{j,p}\}$.
(8) ***Segment-exchange*** (***LLH$_8$***): Suppose $\{O_{1,1}, \cdots, O_{i,1}, \cdots, O_{i+5,1}, \cdots O_{j,1}, \cdots O_{j+5,1}, \cdots O_{n,P_n}\}$ is the operation sequence, select two subsequences $\{O_{i,1}, \cdots, O_{i+5,1}\}$ and $\{O_{j,1}, \cdots O_{j+5,1}\}$ randomly, and exchange them.

Additionally, a simulated annealing (SA) (Lin et al., 2017) approach is implemented into each LLH for further improving the search ability of

```
Procedure Improved LLH
    Set parameter $\xi$, $T_0$ and $T_f$;
    $\partial = \pi$;
    Generate a new sequence $\partial'$ from $\partial$ using a LLH;
    While $T_0 > T_f$
        Generate a new sequence $\partial''$ from $\partial'$ using a LLH;
        If $C_{max}(\partial'') - C_{max}(\partial') < 0$
            $\partial' = \partial''$;
        End if
        $T_0 = T_0 \times \xi$;
    End While
    If $C_{\max}(\partial') - C_{\max}(\pi) < 0$
        $\pi = \partial'$;
    End if
End procedure
```

Fig. 5. Pseudo code of the improved LLH.

QHH. The pseudo code of the improved LLH is shown in Fig. 5, where π denotes an operation vector, ξ is the annealing rate, T_0 and T_f are the initial and final temperatures, respectively.

4.3. State representation

State in RL indicates the current environmental situation after performing a selected action, and it is generally assumed to be observable in a Markov Decision Process. As the SFTSP has a huge search space, the Q-learning needs to visit numerous states if the information about solutions is used to model different states. For this reason, a fitness-based state aggregation technique (Hwang, Lin, Hsu, & Yu, 2011) is employed to separate the state space into several disjoint parts. In QHH, the optimization process is split into a number of episodes to represent different stages. Since solution fitness of each episode has different ranges, the fitness proportion P_t is used to separate the improvement space. The calculation of P_t is given as in Eq. (12), where C_t^{Init} and C_t^{EP} are the solution fitness before and after the search process during an episode period EP, respectively, t is the current episode, $t = 1, 2 \cdots \frac{G}{EP}$, and G is the total evaluation times. In QHH, the proportion is categorized into three aggregate states including [0, 0.85), [0.85, 1) and [1, ∞).

$$P_t = \frac{C_t^{Init}}{C_t^{EP}} \tag{12}$$

4.4. Action selection method

The action selection method aims to balance the exploration and exploitation during the learning process (Falcao, Madureira, & Pereira, 2015). The ε_t-greedy policy is the common method for action selection in Q-learning (Sadhu, Konar, Bhattacharjee, & Das, 2018), which is presented as in Eq. (13), where ε_t is the exploration probability and k is a random number within [0, 1]. In QHH, ε_t is dynamically controlled by Eq. (14), where g_{cur} is the current evaluation times.

$$a_t = \begin{cases} random, & if\ k < \varepsilon_t \\ \underset{a \in A}{\operatorname{argmax}} Q(s_t, a), & otherwise \end{cases} \tag{13}$$

$$\varepsilon_t = 1 - \frac{g_{cur}}{G} \tag{14}$$

4.5. Q-function parameters

Three parameters involved in Q-function (Eq. (9)), which are the discount factor γ, learning rate α_t and reinforcement signal r_t. The discount factor γ indicates the influence of the future reward on the current situation, and its value is highly dependent on problem domain. In QHH, γ needs to be investigated by a set of tuning experiments detailed in Section 5.1.

The learning rate α_t represents the probability of maintaining the existing information or accepting the new information. In QHH, α_t is decreased over evaluation times, and is calculated as in Eq. (15). Obviously, the QHH needs exploration at the initial stage of the search process, and is encouraged to exploit the existing information in the subsequent stages.

$$\alpha_t = 1 - \frac{0.9 \times g_{cur}}{G} \tag{15}$$

$$r_t = \begin{cases} 1, & u < \varepsilon_t \wedge P_f \in [0.85, 1) || u > \varepsilon_t \wedge P_f \in [0, 0.85) \\ 2, & u < \varepsilon_t \wedge P_f \in [0, 0.85) || u > \varepsilon_t \wedge P_f \in [0.85, 1) \\ 0, & otherwise \end{cases} \tag{16}$$

In Q-function, reinforcement signal r_t is used to evaluate the performance of an action. The reward or penalty mechanism used in QHH is

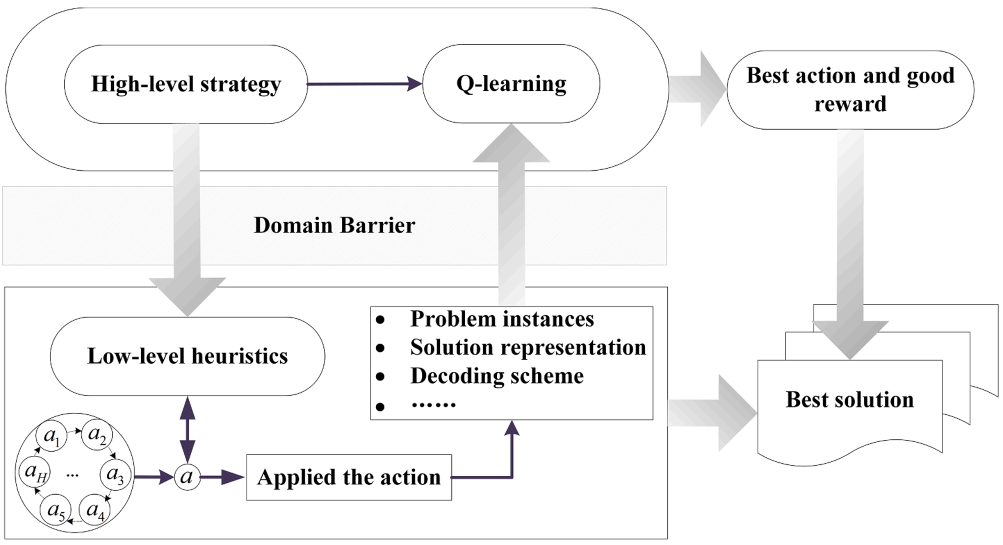

Fig. 6. Framework of the proposed QHH algorithm.

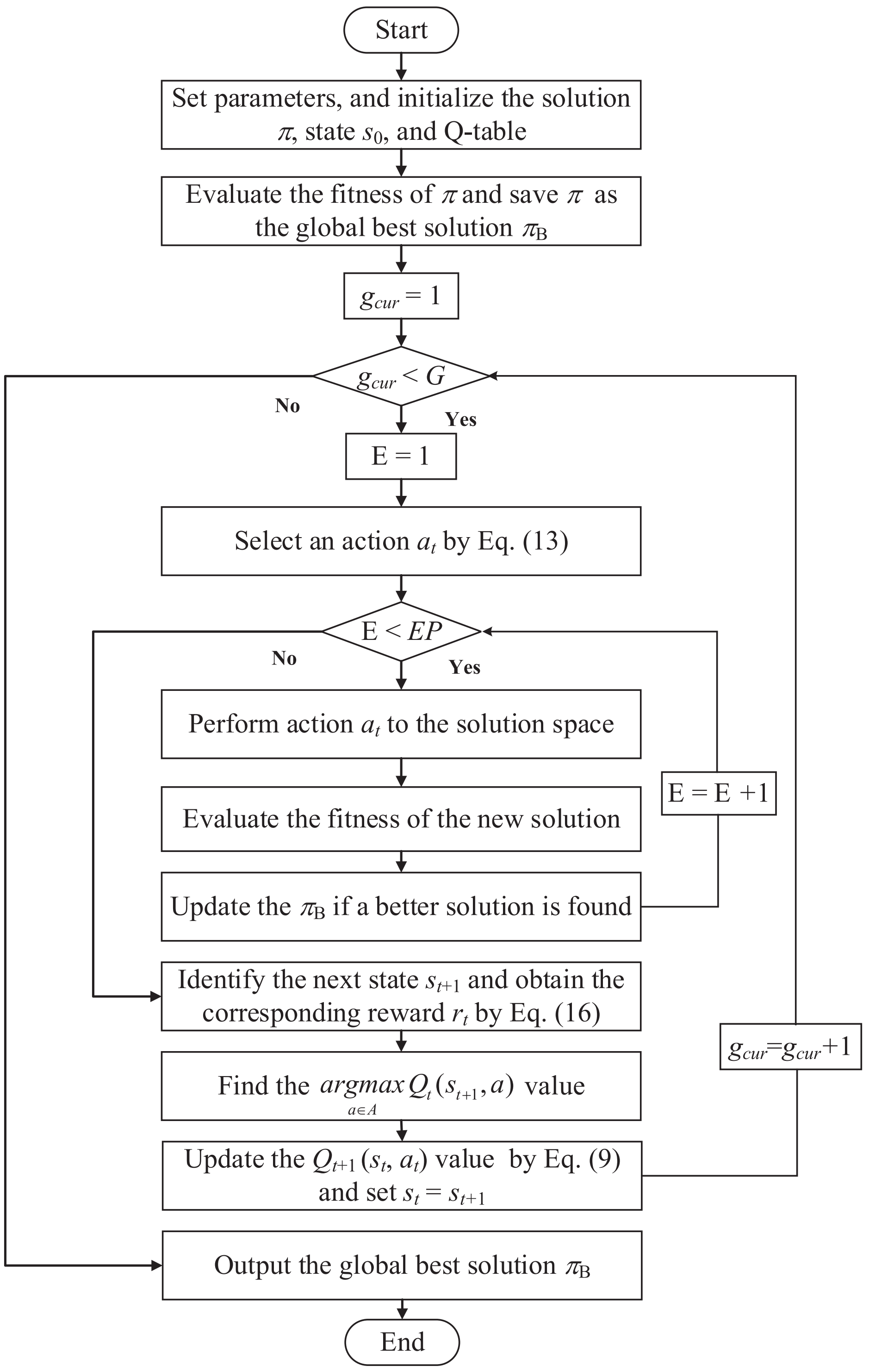
Start
Set parameters, and initialize the solution π, state s_0, and Q-table
Evaluate the fitness of π and save π as the global best solution π_B
$g_{cur} = 1$
$g_{cur} < G$
No
Yes
E = 1
Select an action a_t by Eq. (13)
E < EP
No
Yes
Perform action a_t to the solution space
Evaluate the fitness of the new solution
E = E +1
Update the π_B if a better solution is found
Identify the next state s_{t+1} and obtain the corresponding reward r_t by Eq. (16)
$g_{cur}=g_{cur}+1$
Find the $\underset{a \in A}{argmax}\, Q_t(s_{t+1}, a)$ value
Update the $Q_{t+1}(s_t, a_t)$ value by Eq. (9) and set $s_t = s_{t+1}$
Output the global best solution π_B
End

Fig. 7. Flowchart of the proposed QHH scheme.

presented as in Eq. (16), where u is a random number within [0, 1], and r_t depends on the ε_t probability and the aggregate state after an episode is finished.

4.6. *Algorithmic framework*

The proposed QHH is essentially a single-solution based selection hyper-heuristic, and the solution is represented by an operation sequencing vector associated with an action. Eight simple heuristics are used to construct the LLHs and defined as the executable actions. The Q-learning algorithm is employed as the high-level hyper-heuristic strategy to manipulate a series of LLHs. In addition, solution fitness is evaluated by the decoding scheme after applying action to generate a new solution. The algorithmic framework is presented as in Fig. 6, where the Q-learning algorithm searches the heuristic space directly to achieve a good reward, and the selected action attempts to construct a global best schedule for SFTSP. The flowchart of the proposed QHH is detailed in Fig. 7, and it can be described as follows:

Table 3
Parameters values for each factor level.

Parameter	Factor level			
	1	2	3	4
T_0	2	4	6	8
γ	0.2	0.5	0.7	0.9
EP	2	5	10	15
ξ	0.6	0.7	0.8	0.9

Table 4
Orthogonal array and RV values.

Experiment number	Factor level				RVs	
	T_0	γ	EP	ξ	LS1	WR1
1	1	1	1	1	103.10	251.67
2	1	2	2	2	103.13	250.93
3	1	3	3	3	103.03	250.73
4	1	4	4	4	103.47	249.77
5	2	1	2	3	103.50	253.87
6	2	2	1	4	102.80	253.17
7	2	3	4	1	103.63	250.63
8	2	4	3	2	102.93	249.00
9	3	1	3	4	102.87	253.60
10	3	2	4	3	103.20	251.43
11	3	3	1	2	102.20	250.40
12	3	4	2	1	103.30	251.73
13	4	1	4	2	103.10	253.23
14	4	2	3	1	103.07	252.07
15	4	3	2	4	103.10	253.43
16	4	4	1	3	102.30	254.03

Step 1: Initialize the solution by uniform random distribution, the Q-values of all state-action pairs with zero, and the parameters with the tuned values given in Section 5. 1. The initial P_0 is set to 1, belonging to the aggregate state $[1, \infty)$.
Step 2: Evaluate solution fitness by the decoding scheme, and save the initial solution as the global best solution.
Step 3: Choose an action a_t according to Eq. (13).
Step 4: Perform the action a_t to search the solution space. Update the global best solution when a better one is found. If an EP ends, then calculate the P_t by Eq. (12) and identify the next state s_{t+1}. The corresponding reward r_t is obtained by Eq. (16).
Step 5: Find the $\underset{a\in A}{argmax}Q_t(s_{t+1}, a)$ value, and update the $Q_{t+1}(s_t, a_t)$ value based on Q-function given in Eq. (9). After that, update the current state $s_t = s_{t+1}$.
Step 6: Repeat the procedures from Step 2 to Step 5. Output the best solution if the stopping condition is met.

5. Computational results and comparison

In this section, computational simulations are carried out with a set of benchmark scenarios derived from the practical semiconductor final testing process. The scenarios include 5 large-scale (LS) instances and 5 wide-range (WR) instances. Specifically, the LS scenarios comprise 100 jobs and 36 machines, while the WR scenarios have 60 jobs and 36 machines. All of the instances have defined the resource combinations of machines, available quantities of various resources and setup times. The datasets are available at website http://dalab.ie.nthu.edu.tw/newsen_content.php?id=0. The proposed QHH scheme is coded in Visual C++ 6.0 and performed on a core i5-8265U processor with 1.8 GHz and 8 GB RAM.

5.1. Parameters setting

The discount factor γ, episode period EP, annealing rate ξ and initial temperature T_0 are key parameters to be tested in the proposed QHH algorithm. The Taguchi method of design-of-experiment (DOE) (Montgomery, 2008) is employed to investigate the influence of the parameters on the algorithm performance. As recommended by Hatami, Ruiz, and Andrés-Romano (2015), there would be a big risk of over-fitting when the parameter setting is conducted on the same testing instance. Thus two instances are used for the tuning experiments and each of them is chosen from the LS and WR scenarios, respectively. The first instance is LS1 with 100 jobs and 213 operations, while the second one is WR1 containing 60 jobs and 118 operations. As for the other parameters, the final temperature and the function evaluation times are chosen to be $T_f = 1$ and $G = 10000$, respectively.

Four factor levels are designed for each parameter and their values are listed in Table 3. For each parameter combination, QHH runs independently and repeatedly for 30 times. The average makesapn value is defined as the response variable (RV) here. Apparently, smaller RV value implies better combination of the factor level for parameters. The orthogonal array $L_{16}(4^4)$ and the corresponding RVs are listed in Table 4. Additionally, the obtained RVs are used to carry out the significance test. The F values as well as significance ranks for instances LS1 and WR1 are calculated and presented in Tables 5 and 6, respectively. Trends for each factor level of the parameters for the two instances are shown as in Figs. 8 and 9.

Table 5
RV values and significant rank of LS1.

Factor level	T_0	γ	EP	ξ
1	103.18	103.14	102.60	103.28
2	103.22	103.05	103.26	102.84
3	102.89	102.99	102.98	103.01
4	102.89	103.00	103.35	103.06
F value	0.6350	0.0949	2.2702	0.6373
Rank	3	4	1	2

Table 6
RV values and significant rank of WR1.

Factor level	T_0	γ	EP	ξ
1	250.78	253.09	252.32	251.53
2	251.67	251.90	252.49	250.89
3	251.79	251.30	251.35	252.52
4	253.19	251.13	251.27	252.49
F value	1.3657	1.0776	0.5567	0.8582
Rank	1	2	4	3

It can be drawn from Table 5 that episode period EP is the most significant factor for LS1, followed by ξ and T_0, and γ of the least significance. The best combination for the parameters can be obtained as $T_0 = 6$, $EP = 2$, $\xi = 0.7$ and $\gamma = 0.7$ by Fig. 8. The parameter combinations for the WR instances can be obtained by Fig. 9 similarly and the parameters used in the comparisons are listed in Table 7. We can see from Table 7 and Figs. 8 and 9 that larger T_0 does not necessarily results in better performance since it may cause convergence to local optimum. Moreover, $\xi = 0.7$ is the best option for both instances and too small value for γ leads to poor performance. As for the episode period EP, the corresponding RV varies significantly for the two instances, and $EP = 2$ and $EP = 10$ are preferred by the LS and WR instances, respectively.

5.2. Comparison of QHH to the known results

The performance of the QHH is compared with nFOA (Zheng et al., 2014), HEDA (Wang et al., 2015), CS and CSRS (Cao et al., 2019) in this subsection. For fair comparisons, we re-implemented the above compared algorithms in the same operating environment as the QHH. Two levels of stopping condition are designed for experiments, and the final results are obtained through 10,000 and 50,000 function

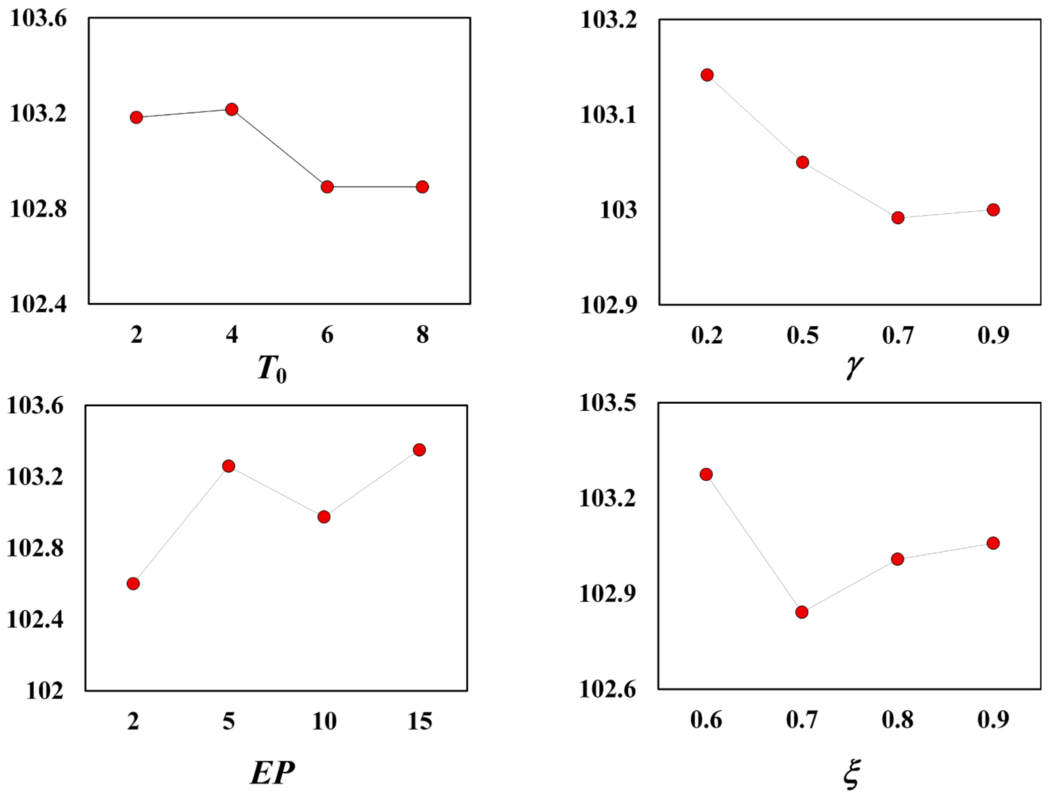

Fig. 8. Trends of factor levels of instance LS1.

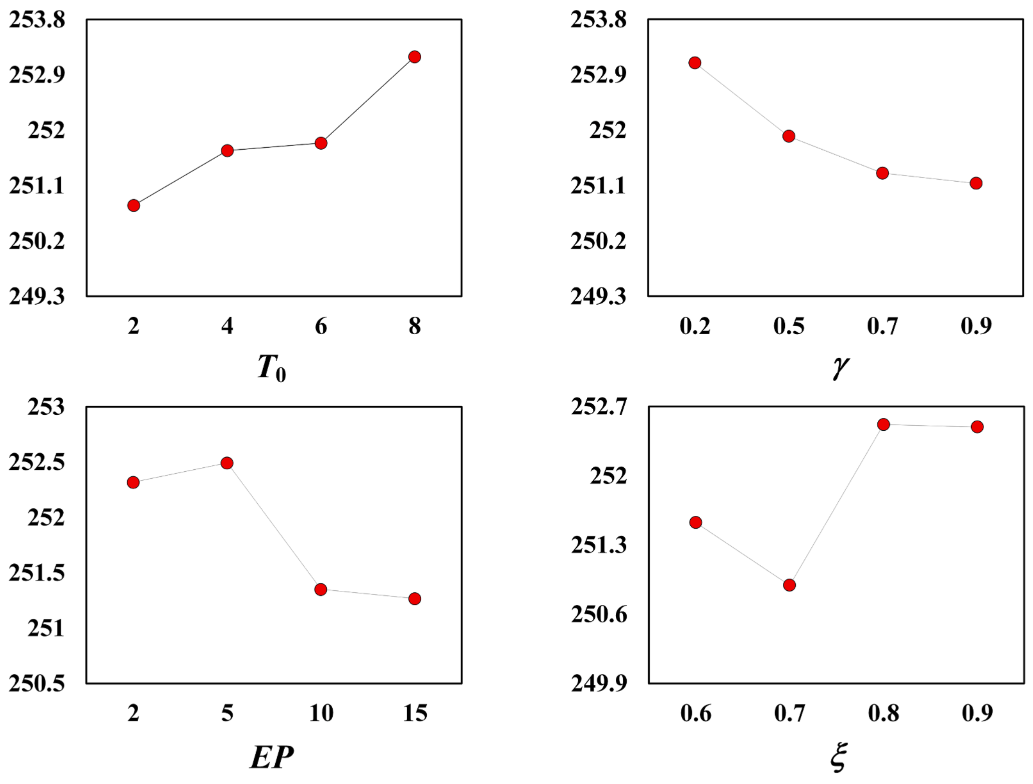

Fig. 9. Trends of factor levels of instance WR1.

evaluation times. As for the other parameters involved in each compared algorithm, they are chosen to be the same values in their experiments. For each scenario, these algorithms are replicated 30 runs independently and the best results are marked in bold.

The computational results of the best and average makespan, and the standard deviation under the two levels of evaluations are listed in Tables 8 and 9 for the compared algorithms, respectively. It is clearly seen that QHH achieves considerably better results in terms of the best and

Table 7
Suggested parameters for different problem instances.

Problem instance	T_0	γ	EP	ξ
Large-scale (LS)	6	0.7	2	0.7
Wide-range (WR)	2	0.9	10	0.7

average makespan than the compared algorithms on most instances. As for the standard deviation, it can be inferred that QHH and HEDA are more stable than the other algorithms since the standard deviation values are relatively smaller in Tables 8 and 9. The above observations prove the superiority of the proposed QHH algorithm. In addition, new best solutions are obtained for LS2-3, LS5, WR2-4, and WR5 with 50,000 evaluations.

In order to statistically verify the performance of the QHH algorithm, a pair-wise *t* test (Boussaïd, Chatterjee, Siarry, & Ahmed-Nacer, 2012) is conducted using the obtained results. Especially, the average makespan values obtained over 30 runs are considered as the samples for the test. However, it is essential to determine whether the samples are normally distributed or not before applying the *t* test. In view of this, two important normality tests, Kolmogorov-Smirnov (K-S) and Shapiro-Wilk (S-W) tests, are used to confirm the normality distribution of the samples. The results of K-S and S-W tests are provided in Tables 10 and 11. It can be seen that the *p*-values generated by the two tests are both >0.05,

Table 8
Computational results of various algorithms on LS1-WR5 with 10,000 evaluations.

Instance	HEDA			nFOA			CS			CSRS			QHH		
	best	avg	std	best	avg	std	best	avg	std	best	avg	std	best	avg	std
LS1	114	117.40	**1.83**	101	105.60	2.43	117	127.00	5.67	103	109.47	3.56	**99**	**101.93**	1.95
LS2	117	120.40	1.94	110	114.07	2.61	134	145.00	7.08	117	123.20	3.38	**107**	**109.93**	**1.80**
LS3	114	116.27	**1.39**	105	107.67	1.42	119	127.43	4.72	107	112.83	3.55	**104**	**106.63**	1.54
LS4	132	135.60	**2.08**	121	125.10	2.09	141	149.97	4.63	125	131.73	3.89	**120**	**122.97**	2.30
LS5	124	127.17	**1.32**	115	118.40	2.14	132	140.33	5.42	116	124.40	4.17	**113**	**116.10**	1.69
WR1	260	267.97	3.92	241	**249.73**	**3.80**	270	295.10	13.97	**240**	261.67	13.02	245	250.20	5.07
WR2	203	210.60	3.51	189	193.73	3.89	205	222.57	7.56	**186**	199.27	5.91	**186**	**191.80**	**2.72**
WR3	234	239.20	3.34	214	222.97	6.94	241	268.73	15.68	217	232.00	12.31	**211**	**216.87**	**3.23**
WR4	**185**	197.80	4.02	**185**	185.80	2.78	193	219.27	14.56	**185**	193.33	11.93	**185**	**185.13**	**0.51**
WR5	236	242.27	**3.51**	213	219.70	4.63	239	262.93	13.30	212	230.67	11.80	**207**	**214.93**	4.70
Average	172	177.47	2.68	159	164.28	3.27	179	195.83	9.26	160	172	7.35	**158**	**161.65**	**2.55**

Table 9
Computational results of various algorithms on LS1-WR5 with 50,000 evaluations.

Instance	HEDA			nFOA			CS			CSRS			QHH		
	best	avg	std	best	avg	std	best	avg	std	best	avg	std	best	avg	std
LS1	111	114.10	1.81	98	104.03	2.43	101	108.63	3.40	**97**	100.83	1.90	**97**	**99.47**	**1.41**
LS2	114	116.80	1.94	108	112.10	2.20	117	124.30	3.83	106	110.47	2.37	**105**	**107.20**	**1.19**
LS3	109	113.63	1.71	103	106.23	1.63	107	111.07	2.79	102	104.93	1.48	**101**	**104.57**	**1.28**
LS4	129	132.07	**1.41**	119	123.90	1.95	125	130.63	3.66	**117**	121.80	2.54	118	**120.17**	1.44
LS5	122	124.27	1.51	113	115.80	1.81	119	122.17	2.29	112	113.83	**1.32**	**110**	**113.73**	1.48
WR1	257	262.20	**2.44**	236	244.57	3.87	243	259.50	7.45	**233**	245.53	4.55	237	**243.57**	3.22
WR2	198	205.43	3.68	186	191.50	4.16	187	197.70	6.19	186	188.43	2.80	**185**	**188.40**	**2.53**
WR3	226	234.13	3.32	212	219.80	5.82	217	231.03	9.20	212	216.10	3.83	**208**	**213.93**	**2.60**
WR4	**185**	189.03	4.15	**185**	185.37	2.01	**185**	193.33	12.15	**185**	185.73	2.79	**185**	**185.00**	**0.00**
WR5	225	232.77	4.94	210	216.53	5.42	208	226.47	9.71	205	210.73	6.06	**201**	**210.40**	**3.64**
Average	168	172.44	2.69	157	161.98	3.13	161	170.48	6.07	156	159.84	2.96	**155**	**158.64**	**1.88**

Table 10
Normality distribution tests of various algorithms with 10,000 evaluations.

Algorithm	K-S test				S-W test			
	Statistic	df	*p*-value	Sig ($p > 0.05$)	Statistic	df	*p*-value	Sig ($p > 0.05$)
HEDA	0.257	10	0.061	**Y**	0.848	10	0.056	**Y**
nFOA	0.259	10	0.056	**Y**	0.859	10	0.074	**Y**
CS	0.26	10	0.055	**Y**	0.864	10	0.086	**Y**
CSRS	0.257	10	0.061	**Y**	0.868	10	0.094	**Y**
QHH	0.256	10	0.062	**Y**	0.864	10	0.085	**Y**

Table 11
Normality distribution tests of various algorithms with 50,000 evaluations.

Algorithm	K-S test				S-W test			
	Statistic	df	*p*-value	Sig ($p > 0.05$)	statistic	df	*p*-value	Sig ($p > 0.05$)
HEDA	0.255	10	0.064	**Y**	0.851	10	0.06	**Y**
nFOA	0.256	10	0.062	**Y**	0.854	10	0.065	**Y**
CS	0.257	10	0.060	**Y**	0.870	10	0.099	**Y**
CSRS	0.256	10	0.062	**Y**	0.862	10	0.081	**Y**
QHH	0.258	10	0.057	**Y**	0.859	10	0.074	**Y**

Table 12
Pair-wise *t* test of various algorithms with 10,000 evaluations.

Algorithms	Mean	SD	SEM	IC-lower	IC-upper	*t*	*p*-value	Sig($p < 0.05$)
HEDA vs QHH	15.818	5.749	1.818	11.706	19.930	8.702	0.000	Y
nFOA vs QHH	2.627	2.023	0.640	1.180	4.074	4.107	0.003	Y
CS vs QHH	34.183	10.768	3.405	26.480	41.886	10.038	0.000	Y
CSRS vs QHH	10.208	3.436	1.087	7.750	12.666	7.270	0.000	Y

Table 13
Pair-wise *t* test of various algorithms with 50,000 evaluations.

Algorithms	Mean	SD	SEM	IC-lower	IC-upper	*t*	*p*-value	Sig($p < 0.05$)
HEDA vs QHH	13.799	5.761	1.822	9.678	17.920	7.575	0.000	Y
nFOA vs QHH	3.339	2.029	0.642	1.888	4.790	5.204	0.001	Y
CS vs QHH	11.839	4.188	1.325	8.843	14.835	8.938	0.000	Y
CSRS vs QHH	1.194	1.069	0.338	0.430	1.958	3.534	0.006	Y

Table 14
CPU time(s) of various algorithms on LS and WR instances with 10,000 evaluations.

Instance	HEDA	nFOA	CS	CSRS	QHH
LS1-5	5.72	1.81	5.67	3.86	**0.71**
WR1-5	2.33	1.64	5.25	3.23	**0.62**
average	4.03	1.73	5.46	3.54	**0.67**

Table 15
CPU time(s) of various algorithms on LS and WR instances with 50,000 evaluations.

Instance	HEDA	nFOA	CS	CSRS	QHH
LS1-5	25.26	5.13	29.14	18.08	**3.17**
WR1-5	9.42	4.24	17.99	14.07	**3.27**
average	17.34	4.68	23.56	16.07	**3.22**

indicating that the employed samples are normally distributed. After that, the two-tailed *t* tests with 95% confidence level are carried out to evaluate the statistical difference among the compared algorithms. Here, the null hypothesis is defined by H_0: $\eta_D = \eta_Q - \eta_O = 0$ stating that there is no significant difference between the average makespan values resulted from applying the QHH and the other algorithms, while the alternative hypothesis is given by H_1: $\eta_D = \eta_Q - \eta_O \neq 0$ associating with the existence of significant difference between the results obtained by the QHH and the other algorithms. The *t* test results are presented in Tables 12 and 13, where SD represents the standard deviation, SEM is the standard error of mean, IC-lower and IC-upper are the confidence interval of the difference, respectively. Obviously, all the *p*-values yielded in Tables 12 and 13 are smaller than 0.05, which means that H_0 is rejected and the QHH is statistically different with the other algorithms at 95% confidence level. Hence, it can be concluded that the proposed hyper-heuristic approach performs much better than the other compared algorithms in solving the SFTSP.

In addition, the average CPU times consumed by various algorithms are also given in Tables 14 and 15 for the comparison of computational efficiency. It can be seen from Tables 14 and 15 that QHH costs much less CPU times than the other compared algorithms. This is quite reasonable since QHH are optimized on a single solution while the others are population-based.

To further illustrate the performance of the QHH, the convergence characteristics for LS1 and WR1 with the 10,000 and 50,000 evaluations are depicted in Figs. 10 and 11, respectively. It can be observed that QHH has faster convergence rate and can find smaller makespan values. Moreover, the quality of the initial solution is better than the compared algorithms, reflecting the efficiency of the proposed solution decoding scheme. The Gantt chart of the new best solutions found by QHH for scenario WR5 is presented in Fig. 12, where the symbol "A" represents the setup time between two adjacent operations on a machine, $O_{i,j}$ denotes the *j*th operation of the *i*th job, and [n1, n2] represents that the starting time and finishing time of the final operation on a machine is n1 and n2, respectively. Obviously, the makespan of the solution in Fig. 12 is 201. The corresponding amounts of remaining resources are presented in Fig. 13. From the figure, it can be seen that the remaining number of resources is always a non-negative value and meet the quantity

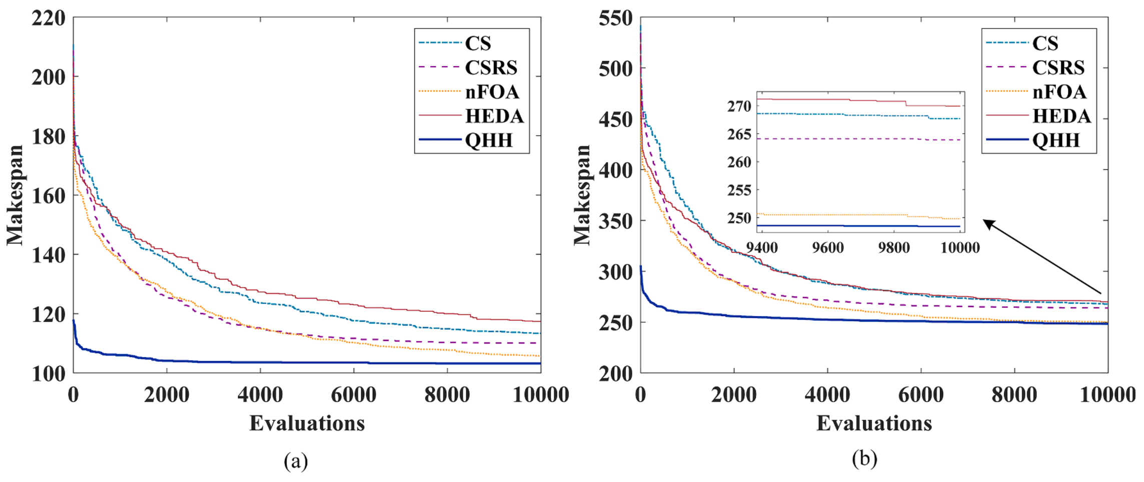

Fig. 10. Average convergence characteristics for instances: (a) LS1 and (b) WR1.

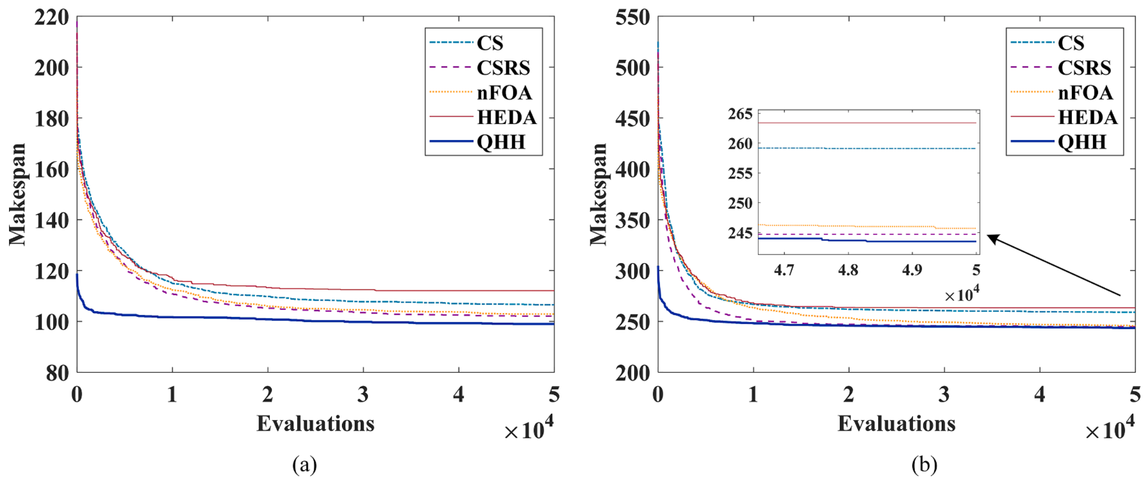

Fig. 11. Average convergence characteristics for the instances: (a) LS1 and (b) WR1.

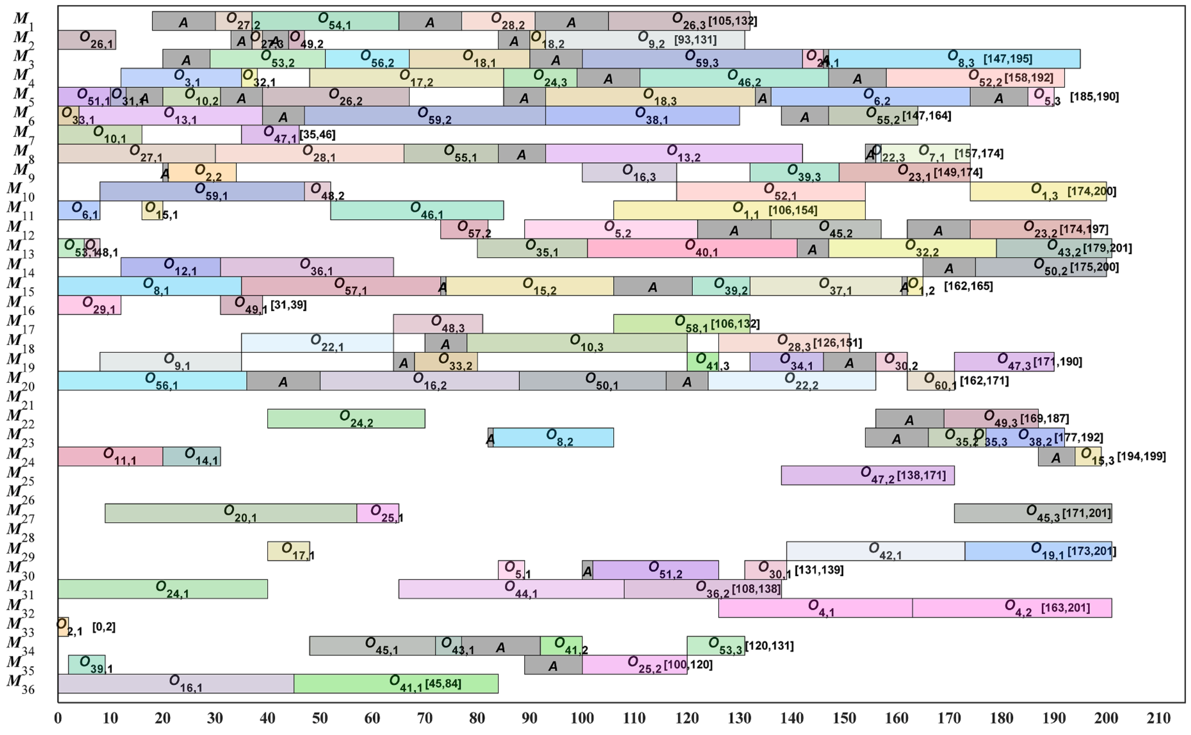

Fig. 12. Gantt chart of the new best solution obtained by QHH for scenario WR5.

limitations, demonstrating the effectiveness of the obtained schedule.

6. Conclusion

In this paper, a QHH algorithm was proposed to solve the SFTSP with the objective of minimizing makespan. An efficient encoding and decoding pair was presented to apply the algorithm and generate feasible schedules. A left-shift scheme was embedded into the decoding process to enhance search ability. Eight simple and efficient heuristics were developed to construct the LLH set and used as a series of actions. The Q-learning algorithm was employed as the high-level heuristic to intelligently select an appropriate action, and then applied the action to solution domain to seek better solutions. The parameters setting were investigated by using the DOE method, and the performance of the QHH was evaluated on the practical instances. Comparative results with the state-of-the-art algorithms demonstrate that the effectiveness of the proposed QHH.

For future work, it will be interesting to extend the SFTSP with multiple criteria or apply the QHH to address some other complicated optimization scheduling problems, such as blocking flowshop, lot-streaming scheduling, dynamic job arrival, etc.

CRediT authorship contribution statement

Jian Lin: Methodology, Formal analysis, Conceptualization, Writing – original draft, Writing - review & editing, Data curation. **Yang-Yuan Li:** Writing - review & editing, Software, Data curation, Investigation, Resources. **Hong-Bo Song:** Conceptualization, Methodology, Writing - review & editing, Supervision, Resources, Investigation.

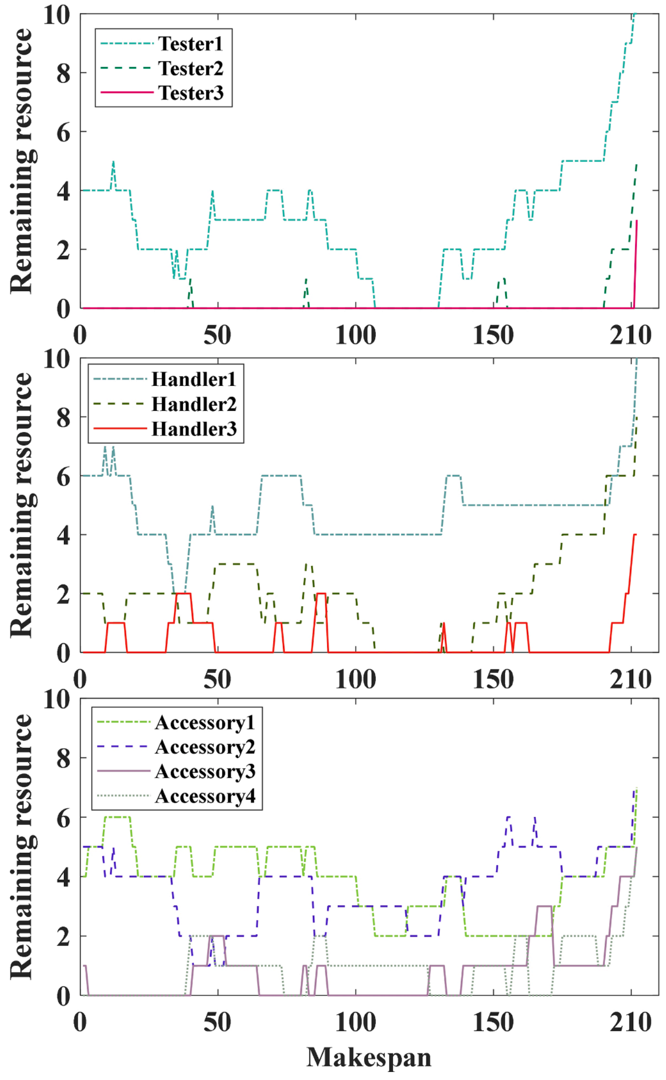

Fig. 13. Amount of remaining resources for scenario WR5.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions. This work is part of a project supported by the National Natural Science Foundation of China (Grant Nos. 61973267, 61503331 and 71671160) and the Zhejiang Provincial Natural Science Foundation of China (Grant Nos. LY19F030007 and LY19G010004).

References

Adriaensen, S., Brys, T., & Nowé, A. (2014). Designing reusable metaheuristic methods: a semi-automated approach. In *2014 IEEE Congress on Evolutionary Computation (CEC)* (pp. 2969–2976). Beijing, China: IEEE.

Asta, S., Özcan, E., & Curtois, T. (2016). A tensor based hyper-heuristic for nurse rostering. *Knowledge-Based Systems, 98*, 185–199.

Boussaïd, I., Chatterjee, A., Siarry, P., & Ahmed-Nacer, M. (2012). Biogeography-based optimization for constrained optimization problems. *Computers & Operations Research, 39*(12), 3293–3304.

Branke, J., Nguyen, S.u., Pickardt, C. W., & Zhang, M. (2016). Automated design of production scheduling heuristics: A review. *IEEE Transactions on Evolutionary Computation, 20*(1), 110–124.

Brucker, P., & Schlie, R. (1990). Job-shop scheduling with multi-purpose machinesScheduling-Probleme in Jop-Shops mit Mehrzweckmaschinen. *Computing, 45*(4), 369–375.

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning, 8*(3-4), 279–292.

Cao, Z., Lin, C., Zhou, M., & Huang, R. (2019). Scheduling semiconductor testing facility by using cuckoo search algorithm with reinforcement learning and surrogate modeling. *IEEE Transactions on Automation Science and Engineering, 16*(2), 825–837.

Chiang, D. M., Guo, R.-S., & Pai, F.-Y. (2008). Improved customer satisfaction with a hybrid dispatching rule in semiconductor back-end factories. *International Journal of Production Research, 46*(17), 4903–4923.

Choong, S. S., Wong, L.-P., & Lim, C. P. (2018). Automatic design of hyper-heuristic based on reinforcement learning. *Information Sciences, 436-437*, 89–107.

Dempster, P., & Drake, J. H. (2016). Two frameworks for cross-domain heuristic and parameter selection using harmony search. *Harmony Search Algorithm, 382*, 83–94.

Drake, J. H., Kheiri, A., Özcan, E., & Burke, E. K. (2020). Recent advances in selection hyper-heuristics. *European Journal of Operational Research, 285*(2), 405–428.

Falcao, D., Madureira, A., & Pereira, I. (2015). Q-learning based hyper-heuristic for scheduling system self-parameterization. In *2015 10th Iberian Conference on Information Systems and Technologies (CISTI)* (pp. 1–7). Aveiro, Portugal: IEEE.

Freed, T., & Leachman, R. C. (1999). Scheduling semiconductor device test operations on multihead testers. *IEEE Transactions on Semiconductor Manufacturing, 12*(4), 523–530.

Gao, Z., Bard, J. F., Chacon, R., & Stuber, J. (2015). An assignment-sequencing methodology for scheduling assembly and test operations with multi-pass requirements. *IIE Transactions, 47*(2), 153–172.

Hao, X.-C., Wu, J.-Z., Chien, C.-F., & Gen, M. (2014). The cooperative estimation of distribution algorithm: A novel approach for semiconductor final test scheduling problems. *Journal of Intelligent Manufacturing, 25*(5), 867–879.

Hatami, S., Ruiz, R., & Andrés-Romano, C. (2015). Heuristics and metaheuristics for the distributed assembly permutation flowshop scheduling problem with sequence dependent setup times. *International Journal of Production Economics, 169*, 76–88.

He, T., Joung, Y. M., Yoon, S. W., Vancheeswaran, R., & Andres, H. R. (2016). Dispatching optimization with sequence dependent setup times in semiconductor final testing scheduling. In Proceedings of the 26th International Conference on Flexibile Automation and Intelligent Manufacturing (FAIM 2016). Seoul, Republic of Korea.

Hsiao, P., Chiang, T., & Fu, L. (2011). A variable neighborhood search-based hyperheuristic for cross-domain optimization problems in CHeSC 2011 competition. In Proceedings of the Fifty-Third Conference of OR Society (OR53). Nottingham, UK.

Hwang, K.-S., Lin, H.-Y., Hsu, Y.-P., & Yu, H.-H. (2011). Self-organizing state aggregation for architecture design of Q-learning. *Information Science, 181*(13), 2813–2822.

Jaradat, M. A. K., Al-Rousan, M., & Quadan, L. (2011). Reinforcement based mobile robot navigation in dynamic environment. *Robotics and Computer-Integrated Manufacturing, 27*(1), 135–149.

Joung, Y. M., He, T., Yoon, S. W., Vancheeswaran, R., Abela, C., & Andres, H. R. (2017). Multi-pass Lot Scheduling Algorithm for Maximizing Throughput at Semiconductor Final Test Facilities. In Proceedings of the 27th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM 2017). Modena, Italy.

Kalender, M., Kheiri, A., Özcan, E., & Burke, E. K. (2013). A greedy gradient-simulated annealing selection hyper-heuristic. *Soft Computing - A Fusion of Foundations, Methodologies & Applications, 17*(12), 2279–2292.

Kheiri, A., & Keedwell, E.d. (2017). A hidden Markov model approach to the problem of heuristic selection in hyper-heuristics with a case study in high school timetabling problems. *Evolutionary Computation, 25*(3), 473–501.

Kiumarsi, B., Lewis, F. L., Modares, H., Karimpour, A., & Naghibi-Sistani, M.-B. (2014). Reinforcement Q-learning for optimal tracking control of linear discrete-time systems with unknown dynamics. *Automatica, 50*, 1167–1175.

Lin, J. (2019). Backtracking search based hyper-heuristic for the flexible job-shop scheduling problem with fuzzy processing time. *Engineering Applications of Artificial Intelligence, 77*, 186–196.

Lin, J., Wang, Z.-J., & Li, X. (2017). A backtracking search hyper-heuristic for the distributed assembly flow-shop scheduling problem. *Swarm and Evolutionary Computation, 36*, 124–135.

Lin, J., Zhu, L., & Gao, K. (2020). A genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem. *Expert Systems with Applications, 140*, 112915. https://doi.org/10.1016/j.eswa.2019.112915

Lin, J. T., Wang, F. K., & Lee, W. T. (2004). Capacity-constrained scheduling for a logic IC final test facility. *International Journal of Production Research, 42*(1), 79–99.

Montgomery, D. C. (2008). *Design and analysis of experiments*. Wiley.

Ovacik, I. M., & Uzsoy, R. (1996). Decomposition methods for scheduling semiconductor testing facilities. *Flexible Services and Manufacturing Journal, 8*, 357–387.

Pearn, W. L., Chung, S. H., Chen, A. Y., & Yang, M. H. (2004). A case study on the multistage IC final testing scheduling problem with reentry. *International Journal of Production Economics, 88*(3), 257–267.

Rajni, & Chana, I. (2014). Bacterial foraging based hyper-heuristic for resource scheduling in grid computing. *Future Generation Computer Systems, 29*(3), 751–762.

Remigio, J. E. J., & Swartz, C. L. E. (2020). Production scheduling in dynamic real-time optimization with closed-loop prediction. *Journal of Process Control, 89*, 95–107.

Ruiz, R., & Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research, 177*(3), 2033–2049.

Sabar, N. R., Ayob, M., Kendall, G., & Rong Qu. (2015). A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems. *IEEE Transactions on Cybernetics, 45*(2), 217–228.

Sadhu, A. K., Konar, A., Bhattacharjee, T., & Das, S. (2018). synergism of firefly algorithm and Q-learning for robot arm path planning. *Swarm and Evolutionary Computation, 43*, 50–68.

Sang, H.-Y., Duan, P.-Y., & Li, J.-Q. (2018). An effective invasive weed optimization algorithm for scheduling semiconductor final testing problem. *Swarm and Evolutionary Computation, 38*, 42–53.

Shen, X.-N., Minku, L. L., Marturi, N., Guo, Y.-N., & Han, Y. (2018). A Q-learning-based memetic algorithm for multi-objective dynamic software project scheduling. *Information Sciences, 428*, 1–29.

Shu, L. (2020). Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. *Applied Soft Computing, 91*, Article 106208.

Uzsoy, R., Church, L. K., Ovacik, I. M., & Hinchman, J. (1992). Dispatching rules for semiconductor testing operations: A computational study. In *Thirteenth IEEE/CHMT International Electronics Manufacturing Technology Symposium* (pp. 272–276). Baltimore, MD, USA: IEEE.

Uzsoy, R., Church, L. K., Ovacik, I. M., & Hinchman, J. (1993). Performance evaluation of dispatching rules for semiconductor testing operations. *Journal of Electronics Manufacturing, 03*(02), 95–105.

Uzsoy, R., Martin-Vega, L. A., Lee, C.-Y., & Leonard, P. A. (1991). Production Scheduling Algorithms for a Semiconductor Test Facility. *IEEE Transactions on Semiconductor Manufacturing, 4*(4), 270–280.

Uzsoy, R., Lee, C.-Y., & Martin-Vega, L. A. (1992). Scheduling semiconductor test operations: Minimizing maximum lateness and number of tardy jobs on a single machine. *Naval Research Logistics, 39*(3), 369–388.

Wang, L., & Zheng, X.-L. (2018). A knowledge-guided multi-objective fruit fly optimization algorithm for the multi-skill resource constrained project scheduling problem. *Swarm and Evolutionary Computation, 38*, 54–63.

Wang, S., & Wang, L. (2015). A knowledge-based multi-agent evolutionary algorithm for semiconductor final testing scheduling problem. *Knowledge-Based Systems, 84*, 1–9.

Wang, S., Wang, L., Liu, M., & Xu, Y.e. (2015). A hybrid estimation of distribution algorithm for the semiconductor final testing scheduling problem. *Journal of Intelligent Manufacturing, 26*(5), 861–871.

Wauters, T., Verbeeck, K., Causmaecker, P. D., & Berghe, G. V. (2013). Boosting metaheuristic search using reinforcement learning. *Hybrid Metaheuristics, 434*, 433–452.

Wei, Q., Liu, D., & Shi, G. (2015). A novel dual iterative Q-learning method for optimal battery management in smart residential environments. *IEEE Transactions on Industrial Electronics, 62*(4), 2509–2518.

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation, 1*(1), 67–82.

Wu, J., & Chien, C. (2008). Modeling semiconductor testing job scheduling and dynamic testing machine configuration. *Expert Systems with Applications, 35*(1-2), 485–496.

Wu, J.-Z., Hao, X.-C., Chien, C.-F., & Gen, M. (2012). A novel bi-vector encoding genetic algorithm for the simultaneous multiple resources scheduling problem. *Journal of Intelligent Manufacturing, 23*(6), 2255–2270.

Zheng, X.-L., Wang, L., & Wang, S.-Y. (2014). A novel fruit fly optimization algorithm for the semiconductor final testing scheduling problem. *Knowledge-Based Systems, 57*, 95–103.