Hindawi Publishing Corporation
Journal of Industrial Engineering
Volume 2013, Article ID 295604, 11 pages
http://dx.doi.org/10.1155/2013/295604

*Research Article*

# Chip Attach Scheduling in Semiconductor Assembly

**Zhicong Zhang, Kaishun Hu, Shuai Li, Huiyu Huang, and Shaoyong Zhao**

*Department of Industrial Engineering, School of Mechanical Engineering, Dongguan University of Technology, Songshan Lake District, Dongguan, Guangdong 523808, China*

Correspondence should be addressed to Zhicong Zhang; stephen1998@gmail.com

Received 13 December 2012; Accepted 20 February 2013

Academic Editor: Josefa Mula

Chip attach is the bottleneck operation in semiconductor assembly. Chip attach scheduling is in nature unrelated parallel machine scheduling considering practical issues, for example, machine-job qualification, sequence-dependant setup times, initial machine status, and engineering time. The major scheduling objective is to minimize the total weighted unsatisfied Target Production Volume in the schedule horizon. To apply $Q$-learning algorithm, the scheduling problem is converted into reinforcement learning problem by constructing elaborate system state representation, actions, and reward function. We select five heuristics as actions and prove the equivalence of reward function and the scheduling objective function. We also conduct experiments with industrial datasets to compare the $Q$-learning algorithm, five action heuristics, and Largest Weight First (LWF) heuristics used in industry. Experiment results show that $Q$-learning is remarkably superior to the six heuristics. Compared with LWF, $Q$-learning reduces three performance measures, objective function value, unsatisfied Target Production Volume index, and unsatisfied job type index, by considerable amounts of 80.92%, 52.20%, and 31.81%, respectively.

## 1. Introduction

Semiconductor manufacturing consists of four basic steps: wafer fabrication, wafer sort, assembly, and test. Assembly and test are back-end steps. Semiconductor assembly contains many operations, such as reflow, wafer mount, saw, chip attach, deflux, EPOXY, cure, and PEVI. IS factory is a site for back-end semiconductor manufacturing where chip attach is the bottleneck operation in the assembly line. In terms of Theory of Constraints (TOC), the capacity of a shop floor depends on the capacity of the bottleneck, and a bottleneck operation gives a tremendous impact upon the performance of the whole shop floor. Consequently, scheduling of chip attach station has a significant effect on the performance of the assembly line. Chip attach is performed in a station which consists of ten parallel machines; thus, chip attach scheduling in nature is some form of unrelated parallel machine scheduling under certain realistic restrictions.

Research on unrelated parallel machine scheduling focuses on two sorts of criteria: completion time or flow time related criteria and due date related criteria. Weng et al. [1] proposed a heuristic algorithm called "Algorithm 9" to minimize the total weighted completion time with setup consideration. Algorithm 9 was demonstrated to be superior to six heuristic algorithms. Gairing et al. [2] presented an effective combinatorial approximate algorithm for makespan objective. Mosheiov [3] and Mosheiov and Sidney [4] converted an unrelated parallel machine scheduling problem with total flow time objective into polynomial number of assignment problems. The scheduling problem was tackled by solving the derived assignment problems. Yu et al. [5] formulated unrelated parallel machine scheduling problems as mixed integer programming and dealt with them using Lagrangian Relaxation. They examined six measures such as makespan and mean flow time. Promising results were achieved compared with a modified FIFO method.

Besides completion time or flow time related criteria, tardiness objectives are also employed frequently. Dispatching rules are widely applied to production scheduling with a tardiness objective, such as Earliest Due Date (EDD), Shortest Processing Time (SPT), Critical Ratio (CR), Minimal Slack (MS), Modified Due Date (MDD) [6, 7], Apparent Tardiness

Cost (ATC) [8, 9], and COVERT [10–12]. More complicated heuristic algorithms and local search methods are also developed. Bank and Werner [13] addressed the problem of minimizing the weighted sum of linear earliness and tardiness penalties in unrelated parallel machine scheduling. They derived some structural properties useful to searching for an approximate solution and proposed various constructive and iterative heuristic algorithms. Liaw et al. [14] found the efficient lower and upper bounds of minimizing the total weighted tardiness by a two-phase heuristics based on the solution to an assignment problem. They also presented a branch-and-bound algorithm incorporating various dominance rules. Kim et al. [15] studied batch scheduling of unrelated parallel machines with a total weighted tardiness objective and setup times consideration. They examined four search heuristics for this problem: the earliest weighted due date, the shortest weighted processing time, the two-level batch scheduling heuristic, and the simulated annealing method.

We are concerned in the paper about a particular Target Production Volume (TPV) oriented optimization objective. In real production in IS factory, the planning department figures out the TPV of each job type on chip attach operation in a schedule horizon. Thus, the major objective of chip attach scheduling is to meet TPVs to the fullest extent (see Section 2.1 for details). We apply reinforcement learning (RL), an artificial intelligence method, for this study. We first present a brief concept of reinforcement learning.

*1.1. Q-Learning.* Reinforcement learning is a machine learning method proposed to approximately solve large-scale Markov Decision Process (MDP) or Semi-Markov Decision Process (SMDP) problems. Reinforcement learning problem is a model in which an agent learns to select optimal or near-optimal actions for achieving its long-term goals (to maximize the total or average reward) through trial-and-error interactions with dynamic environment. In this paper, we address RL problems of episodic task, that is, problems with a terminal state. Sutton and Barto [16] defined four key elements of RL algorithms: policy, reward function, value function, and model of the environment. A policy determines the agent's action at each state. A reward function determines the payment on transition from one state to another. A value function specifies the value of a state or a state-action pair in the long run, the expected total reward for an episode. By learning from interaction between the agent and its environment, value-based RL algorithms aim to approximate the optimal state or action value function through iteration and thus find a near-optimal policy. Compared with dynamic programming, RL algorithms do not need to know the transition probability and reduce the computational effort.

$Q$-learning is one of the most widely applied RL algorithms based on value iteration. $Q$-learning was first proposed by Watkins [17]. Convergence results of tabular $Q$-learning were obtained by Watkins and Dayan [18], Jaakkola et al. [19], and Tsitsiklis [20]. Bertsekas and Tsitsiklis [21] demonstrated that $Q$-learning produces the optimal policy in discounted reward problems under certain conditions. $Q$-learning uses $Q(s,a)$, called $Q$-value, to represent the value of a state-action pair. $Q(s,a)$ is defined as follows:

$$Q(s,a) = \sum_{s' \in S} p(s,a,s') \left[ r(s,a,s') + \gamma V^{*}(s') \right], \tag{1}$$

where $S$ denotes the state space, $p(s,a,s')$ denotes the transition probability from $s$ to $s'$ taking action a, $r(s,a,s')$ denotes the reward on transition from $s$ to $s'$ taking action a, $\gamma$ $(0 < \gamma \leq 1)$ is a discounted factor, and $V^{*}(\cdot)$ is the optimal state value function.

In terms of Bellman optimality function, the following holds for arbitrary $s \in S$, where $A(s)$ denotes the set of actions available for state $s$:

$$V^{*}(s) = \max_{a \in A(s)} Q(s,a). \tag{2}$$

From (1) and (2), the following equation holds:

$$Q(s,a) = \sum_{s' \in S} p(s,a,s') \left[ r(s,a,s') + \gamma \max_{a' \in A(s')} Q(s',a') \right] \quad \forall (s,a). \tag{3}$$

Equation (3) is the basic transformation of $Q$-learning algorithm. The step-size version of $Q$-learning is

$$Q(s,a) = Q(s,a) + \alpha \left[ r(s,a,s') + \gamma \max_{a' \in A(s')} Q(s',a') - Q(s,a) \right], \quad \forall (s,a), \tag{4}$$

where $\alpha$ $(0 < \alpha \leq 1)$ is learning rate. Using historical samples or simulation experiments, $Q$-learning obtains a near-optimal policy by driving action-value function, $Q(s,a)$, towards the optimal action-value function, $Q^{*}(s,a)$, through iteration based on formula (4).

Recently, RL has drawn attention from production scheduling. S. Riedmiller and M. Riedmiller [22] used $Q$-learning to solve stochastic and dynamic job shop scheduling problem with the overall tardiness objective. Some typical heuristic dispatching rules, SPT, LPT, EDD, and MS, were chosen as actions and compared with the $Q$-learning method. Aydin and Öztemel [23] applied a $Q$-learning algorithm to minimize the mean tardiness of dynamic job shop scheduling. Their results showed that the RL-scheduling system outperformed the use of each of the three rules (SPT, COVERT, and CR) individually with mean tardiness objective in most of the testing cases. Hong and Prabhu [24] formulated setup minimization problem (minimizing the sum of due date deviation and setup cost) in JIT manufacturing systems as an SMDP and solved it by tabular $Q$-learning method. Experiment results showed that $Q$-learning algorithms achieved significant performance improvement over usual dispatching rules such as EDD in complex real-time shop floor control problems for JIT production. Wang and Usher [25] applied $Q$-learning to select dispatching rules for the single machine scheduling problem. Csáji et al. [26] proposed an adaptive iterative distributed scheduling algorithm operated in a market-based production control system, where every machine and job is associated with its own software agent. Singh et al. [27] proposed an online reinforcement learning algorithm for call admission control. The approach optimized the SMDP performance criterion with respect to a family of parameterized policies. Multi-agent reinforcement learning system has also been applied to scheduling or control problems, for example, Kaya and Alhajj [28], Paternina-Arboleda and Das [29], Mariano-Romero et al. [30], Vengerov [31], Iwamura et al. [32].

Applications of RL algorithms to scheduling problems have not been thoroughly explored in the prior studies. In this study, we employ $Q$-learning algorithm to resolve chip attach scheduling problem and achieve overwhelming experimental results compared with six heuristic algorithms. The remainder of this paper is organized as follows. We describe the problem and convert it into RL problem explicitly in Section 2, present the RL algorithm in Section 3, conduct the computational experiments and analysis in Section 4, and draw conclusions in Section 5.

## 2. RL Formulation

*2.1. Problem Statement.* The scheduling problem concerned in this paper is described as follows. The work station for chip attach operation consists of $m$ parallel machines and processes $n$ types of jobs. The bigger the weight of a job type is, the more important it is. Each job needs to be processed on one machine only and one machine processes at most one job at a time. Any job type (say, $j$) is only allowed to be processed on subset $M_j$ of the $m$ parallel machines. The jobs of the same type $j$ have a deterministic processing time $p_{i,j}$ ($1 \le i \le m$; $1 \le j \le n$) if they are processed on machine $i$. The machines are unrelated; that is, $p_{i,j}$ is independent of $p_{k,j}$ for all jobs $j$ and all machines $i \ne k$. The production is lot based. Normally, one lot contains more than 1000 units. Thus, the processing time is the time for processing one lot and processing is nonpreemptive (i.e., once a machine starts processing one lot, it cannot process another one until it completely processes this lot). Setup time between job type $j1$ and $j2$ is $s_{j1,j2}$ ($1 \le j1, j2 \le n$). The setup times are deterministic and sequence dependant. Trivially, $s_{j,j} = 0$ holds for arbitrary $j$ ($1 \le j \le n$) and $s_{j,x} + s_{x,q} > s_{j,q}$ holds for arbitrary $j, x, q$ ($1 \le j, x, q \le n$).

The usage of a machine is considered to be in one of two categories: engineering time (e.g., maintenance time) and production time. We only need to schedule the production in production time, the total available time in a schedule horizon deducting the engineering time. Production time is divided into initial production time and normal production time. We consider the initial machine status in the schedule horizon. If a machine is processing a lot, called "initial lot," at the beginning of a schedule horizon, it is not allowed to process any other lot until it completely processes the remaining units in the initial lot (called initial volume). The time for processing the unprocessed initial volume in the initial lot is called "initial production time." Since the production of nonbottleneck operations is determined by the bottleneck operation, we assume that the jobs are always available for processing on chip attach operation when they are needed.

The primary objective of chip attach scheduling is to minimize the total weighted unsatisfied TPV of a schedule horizon. Since equipment of semiconductor manufacturing is very expensive, machine utilization should be kept in a high level. Hence, on the premise that TPVs of all job types are entirely satisfied, the secondary objective of chip attach scheduling is to process as much as weighted excess volume to relieve the burden of the next schedule horizon. The objective function is formulated as follows:

$$\min \sum_{j=1}^{n} w_j \left(D_j - Y_j\right)^{+} - \sum_{j=1}^{n} \frac{w_j}{M} \left(Y_j - D_j\right)^{+}, \qquad (5)$$

where $w_j$ ($1 \le j \le n$) is the weight per unit of job type $j$, $D_j$ ($1 \le j \le n$) is the predetermined TPV of job type $j$ (including the initial volume in the initial lots), and $Y_j$ ($1 \le j \le n$) is the processed volume of job type $j$. $D_j$ can be represented as follows:

$$D_j = \sum_{i=1}^{m} \omega(i, j) I_i + k_j L \quad \left(k_j = 0, 1, \dots\right), \qquad (6)$$

where $I_i$ denotes the initial volume in the initial lot processed by machine $i$ at the beginning of the schedule horizon, $L$ is lot size, and

$$\omega(i, j) = \begin{cases} 1, & \text{if machine } i \text{ is processing job type } j \\ & \text{in the beginning of the schedule horizon,} \\ 0, & \text{otherwise.} \end{cases} \qquad (7)$$

Calculation of $Y_j$ is rate based, interpreted as follows. Suppose machine $i$ processes lot $LQ$ (belonging to job type $q$), proceeding lot $LJ$ (belonging to job type $j$). Let $ts_{LJ}$ denote the start time of setup for $LJ$; then, the completion time of $LJ$ is $ts_{LJ} + s_{q,j} + p_{i,j}$. Let $\Delta Y_{i,j}(t)$ denote the increase of processed volume of job type $j$ because of processing $LJ$ on machine $i$ from time $ts_{LJ}$ to $t$, defined as follows:

$$\Delta Y_{i,j}(t) = \frac{(t - ts_{LJ}) L}{s_{q,j} + p_{i,j}} \quad \left(ts_{LJ} \le t \le ts_{LJ} + s_{q,j} + p_{i,j}\right). \qquad (8)$$

$M$ is a positive number which is large enough. $M$ is set following the next inequality:

$$M > \max \left\{ \frac{\left(w_q + w_x\right)\left(s_{v,j} + p_{i,j}\right)}{w_j p_{i,q}}, \frac{w_q \left(s_{v,x} + p_{i,x}\right)\left(s_{x,j} + p_{i,j}\right)}{p_{i,j} \min_{1 \le c \le m, 1 \le a,b,k \le n} \left\{w_k \left(s_{a,b} + p_{c,b}\right)\right\}} \right\}$$
$$\left(\forall 1 \le i \le m, 1 \le j, q, v, x \le n\right). \qquad (9)$$

For an optimal schedule minimizing objective function (5), if (9) holds and there exists $j$ $(1 \le j \le n)$ such that $Y_j > D_j$, then

$$Y_j + \sum_{i=1}^{m} \beta(i,j) U_i \ge D_j \quad (\forall 1 \le j \le n), \tag{10}$$

where $U_i$ denotes the unprocessed volume in the last lot processed by machine $i$ at the end of this schedule horizon (i.e., the initial volume of the next schedule horizon) and

$$\beta(i,j) = \begin{cases} 1, & \text{if machine } i \text{ is processing job type } j \text{ at the end of the schedule horizon,} \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

According to inequality (9), in any schedule minimizing objective function (5), any machine will not process a lot belonging to a job type whose TPV has been satisfied until TPV of any other job types is also fully satisfied. In other words, inequality (9) guarantees that the objective function takes minimization of the total weighted unsatisfied TPV (the first item of objective function (5)) as the first priority. The fundamental problem in applying reinforcement learning to scheduling is to convert scheduling problems into RL problems, including representation of state, construction of actions, and definition of reward function.

*2.2. State Representation and Transition Probability.* We first define the state variables. State variables describe the major characteristics of the system and are capable of tracking the change of the system status. The system state can be represented by the vector

$$\varphi = \left[T_i^0 (1 \le i \le m); T_i (1 \le i \le m); t_i (1 \le i \le m); d_j (1 \le j \le n); e_i (1 \le i \le m)\right], \tag{12}$$

where $T_i^0$ $(1 \le i \le m)$ denotes the job type of which the latest lot completely processed on machine $i$, $T_i$ $(1 \le i \le m)$ denotes the job type of which the lot being processed on machine $i$ ($T_i$ equals zero if machine $i$ is idle), $t_i$ $(1 \le i \le m)$, denotes the time starting from the beginning of the latest setup on machine $i$ (for convenience, we assume that there is a zero-time setup if $T_i^0 = T_i$), $d_j$ $(1 \le j \le n)$ is unsatisfied TPV (i.e., $(D_j - Y_j)^+$), and $e_i$ $(1 \le i \le m)$ represents the unscheduled normal production time of machine $i$.

Considering the initial status of machines, the initial system state of the schedule horizon is

$$s_0 = \left[T_{i,0}^0 (1 \le i \le m); T_{i,0} (1 \le i \le m); t_{i,0} (1 \le i \le m); D_j (1 \le j \le n); \mathrm{TH} - \sigma_i - \mathrm{TE}_i (1 \le i \le m)\right], \tag{13}$$

where TH denotes the overall available time in the schedule horizon, $\sigma_i$ denotes the initial production time of machine $i$, and $\mathrm{TE}_i$ denotes the engineering time of machine $i$.

There are two kinds of events triggering state transitions: (1) completion of processing a lot on one or more machines; (2) any machine's normal production time is entirely scheduled. If the triggering event is completion of processing, the state at the decision-making epoch is represented as

$$s_d = \left[T_{i,d}^0 (1 \le i \le m); T_{i,d} (1 \le i \le m); t_{i,d} (1 \le i \le m); d_{j,d} (1 \le j \le n); e_{i,d} (1 \le i \le m)\right], \tag{14}$$

where $\{i \mid T_{i,d} = 0, 1 \le i \le m\} \ne \Phi$. If $T_{i,d} = 0$ (machine $i$ is idle), then $t_{i,d} = 0$. If the triggering event is using up a machine's normal production time, then $\{i \mid e_{i,d} = 0, 1 \le i \le m\} \ne \Phi$.

Assume that after taking action $a$, the system state immediately transfers form $s_d$ to an interim state, $s$, as follows:

$$s = \left[T_i^0 (1 \le i \le m); T_i (1 \le i \le m); t_i (1 \le i \le m); d_j (1 \le j \le n); e_i (1 \le i \le m)\right], \tag{15}$$

where $T_i > 0$ for all $i$ $(1 \le i \le m)$; that is, all machines are busy.

Let $\Delta t$ denote the sojourn time at state $s$; then, $\Delta t = \min\{\min_{1 \le i \le m}\{s_{T_i^0, T_i} + p_{i,T_i} - t_i\}, \min_{1 \le i \le m}\{e_i \mid e_i > 0\}\}$. Let $\Lambda = \{i \mid s_{T_i^0, T_i} + p_{i,T_i} - t_i = \Delta t\}$; then, the state at the next decision-making epoch is represented as

$$\begin{aligned} s' = \Big[ & T_i (i \in \Lambda), T_i^0 (i \notin \Lambda); T_i = 0 (i \in \Lambda), T_i (i \notin \Lambda); \\ & 0 (i \in \Lambda), t_i + \Delta t (i \notin \Lambda); \\ & d_j - \frac{\Delta t L \sum_{i=1}^{m} \delta_Y (T_i, j)}{s_{T_i^0, j} + p_{i,j}} (1 \le j \le n); \\ & \max\{e_i - \Delta t, 0\} (1 \le i \le m)\Big], \end{aligned} \tag{16}$$

where

$$\delta_Y (T_i, j) = \begin{cases} 1, & \text{if } T_i = j, \\ 0, & \text{if } T_i \ne j. \end{cases} \tag{17}$$

Apparently we have $P_{s_d,s'}^a = 1$, where $P_{s_d,s'}^a$ denotes the one-step transition probability from state $s_d$ to state $s'$ under action $a$. Let $s_u$ and $\tau_u$ denote the system state and time, respectively, at the $u$th decision-making epoch. It is easy to show that

$$\begin{aligned} & P\{s_{u+1} = X, \tau_{u+1} - \tau_u \le t \mid s_0, s_1, \ldots, s_u; \tau_0, \tau_1, \ldots, \tau_u\} \\ & \quad = P\{s_{u+1} = X, \tau_{u+1} - \tau_u \le t \mid s_u; \tau_u\}, \end{aligned} \tag{18}$$

where $\tau_{u+1} - \tau_u$ is the sojourn time at state $s_u$. That is, the decision process associated with $(s, \tau)$ is a Semi-Markov Decision Process with particular transition probability and sojourn times. The terminal state of an episode is

$$s_e = \left[T_{i,e}^0 (1 \le i \le m); T_{i,e} (1 \le i \le m); t_{i,e} (1 \le i \le m); d_{j,e} (1 \le j \le n); 0 (1 \le i \le m)\right]. \tag{19}$$

*2.3. Action.* Prior domain knowledge can be utilized to fully exploit the agent's learning ability. Apparently, an optimal schedule must be nonidle (i.e., any machine has no idle time during the whole schedule). It may happen that more than one machine are free at the same decision-making epoch. An action determines which lot to be processed on which machine. In the following, we define seven actions using heuristic algorithms.

*Action 1.* Select jobs by WSPT heuristics as follows.

*Algorithm 1.* WSPT heuristics.

*Step 1.* Let SM denote the set of free machines at a decision-making epoch.

*Step 2.* Choose machine $k$ to process job type $q$, with $(k, q) = \text{argmin}_{(i,j)}\{(s_{T_i^0,j} + p_{i,j})/w_j \mid 1 \le j \le n, i \in M_j \text{ and } i \in \text{SM}\}$.

*Step 3.* Remove $k$ from SM. If $\text{SM} \ne \Phi$, go to Step 2; otherwise, the algorithm halts.

*Action 2.* Select jobs by MWSPT (modified WSPT) heuristics as follows.

*Algorithm 2.* MWSPT heuristics.

*Step 1.* Define SM as Step 1 in Algorithm 1, and let SJ denote the set of job types whose TPVs have not been satisfied at a decision-making epoch; that is, $\text{SJ} = \{j \mid Y_j < D_j, 1 \le j \le n\}$. If $\text{SJ} = \Phi$, go to Step 4.

*Step 2.* Choose job type $q$ to process on machine $k$, with $(k, q) = \text{argmin}_{(i,j)}\{(s_{T_i^0,j} + p_{i,j})/w_j \mid j \in \text{SJ}, i \in M_j \text{ and } i \in \text{SM}\}$.

*Step 3.* Remove $k$ from SM. Set $Y_q = Y_q + L$ and update SJ. If $\text{SJ} \ne \Phi$ and $\text{SM} \ne \Phi$, go to Step 2; if $\text{SJ} = \Phi$ and $\text{SM} \ne \Phi$, go to Step 4; otherwise, the algorithm halts.

*Step 4.* Choose machine $k$ to process job type $q$, with $(k, q) = \text{argmin}_{(i,j)}\{(s_{T_i^0,j} + p_{i,j})/w_j \mid 1 \le j \le n, i \in M_j \text{ and } i \in \text{SM}\}$.

*Step 5.* Remove $k$ from SM. If $\text{SM} \ne \Phi$, go to Step 4; otherwise, the algorithm halts.

*Action 3.* Select jobs by Ranking Algorithm (RA) as follows.

*Algorithm 3.* Ranking Algorithm.

*Step 1.* Define SM and SJ as Step 1 in Algorithm 2. If $\text{SJ} = \Phi$, go to Step 5.

*Step 2.* For each job type $j$ $(j \in \text{SJ})$, sort the machines in increasing order of $(s_{V_i,j} + p_{i,j})$ $(1 \le i \le m)$, where $V_i$ is defined as follows.

$$V_i = \begin{cases} T_i, & \text{if machine } i \text{ is busy} \\ T_i^0, & \text{if machine } i \text{ is free} \end{cases} \quad (1 \le i \le m). \qquad (20)$$

Let $g_{i,j}$ $(1 \le g_{i,j} \le m)$ denote the order of machine $i$ $(1 \le i \le m)$ for job type $j$ $(1 \le j \le n)$.

*Step 3.* Choose job $q$ to process on machine $k$, with $(k, q) = \text{argmin}_{(i,j)}\{g_{i,j} \mid j \in \text{SJ}, i \in M_j \text{ and } i \in \text{SM}\}$. If there exist two or more machine-job combinations (say, machine-job combination $(i_1, j_1)$, $(i_2, j_2)$, $\dots$, $(i_h, j_h)$) with the same minimal order; that is, $(i_e, j_e) = \text{argmin}_{(i,j)}\{g_{i,j} \mid j \in \text{SJ}, i \in M_j \text{ and } i \in \text{SM}\}$ holds for $e$ $(1 \le e \le h)$, then choose job type $j_e$ to process on machine $i_e$, with $(i_e, j_e) = \text{argmin}_{(i,j)}\{(s_{Vi_e,j_e} + p_{i_e,j_e})/w_{j_e} \mid 1 \le e \le h\}$.

*Step 4.* Remove $k$ or $i_e$ from SM. Set $Y_q = Y_q + L$ or $Y_{j_e} = Y_{j_e} + L$ and update SJ. If $\text{SJ} \ne \Phi$ and $\text{SM} \ne \Phi$, go to Step 3; if $\text{SJ} = \Phi$ and $\text{SM} \ne \Phi$, go to Step 5; otherwise, the algorithm halts.

*Step 5.* Choose job $q$ to process on machine $k$, with $(k, q) = \text{argmin}_{(i,j)}\{g_{i,j} \mid 1 \le j \le n, i \in M_j \text{ and } i \in \text{SM}\}$. If there exist two or more machine-job combinations (say, machine-job combinations $(i_1, j_1)$, $(i_2, j_2)$,..., $(i_h, j_h)$) with the same minimal order, choose job type $j_e$ to process on machine $i_e$, with $(i_e, j_e) = \text{argmin}_{(i,j)}\{(s_{Vi_e,j_e} + p_{i_e,j_e})/w_{j_e} \mid 1 \le e \le h\}$.

*Step 6.* Remove $k$ or $i_e$ from SM. If $\text{SM} \ne \Phi$, go to Step 5; otherwise, the algorithm halts.

*Action 4.* Select jobs by LFM-MWSPT heuristics as follows.

*Algorithm 4.* LFM-MWSPT heuristics.

*Step 1.* Define SM and SJ as Step 1 in Algorithm 2.

*Step 2.* Select a free machine (say, $k$) from SM by LFM (Least Flexible Machine; see [33]) rule and choose a job type to process on machine $k$ following MWSPT heuristics.

*Step 3.* Remove $k$ from SM. If $\text{SM} \ne \Phi$, go to Step 2; otherwise, the algorithm halts.

*Action 5.* Select jobs by LFM-RA heuristics as follows.

*Algorithm 5.* LFM-RA heuristics.

*Step 1.* Define SM and SJ as Step 1 in Algorithm 2.

*Step 2.* Select a free machine (say, $k$) from SM by LFM rule and choose a job type to process on machine $k$ following Ranking Algorithm.

*Step 3.* Remove $k$ from SM. If $\text{SM} \ne \Phi$, go to Step 2; otherwise, the algorithm halts.

*Action 6.* Each free machine selects the same job type as the latest one it processed.

*Action 7.* Select no job.

At the start of a schedule horizon, the system is at initial state $s_0$. If there are free machines, they select jobs to process by taking one of Actions 1–6; otherwise, Action 7 is chosen. Afterwards, when any machine completes processing a lot or any machine's normal production time is completely scheduled, the system transfers into a new state, $s_u$. The agent selects an action at this decision-making epoch and the system state transfers into an interim state, $s$. When, again, any machine completes processing a lot or any machine's normal production time used is up, the system transfers into the next decision-making state $s_{u+1}$ and the agent receive reward $r_{u+1}$, which is computed due to $s_u$ and the sojourn time between the two transitions into $s_u$ and $s_{u+1}$ (as shown in Section 2.4). The previous procedure is repeated until a terminal state is attained. An episode is a trajectory from the initial state to a terminal state of a schedule horizon. Action 7 is available only at the decision-making states when all machines are busy.

*2.4. Reward Function.* A reward function follows several disciplines. It indicates the instant impact of an action on the schedule, that is, to link the action with immediate reward. Moreover, the accumulated reward indicates the objective function value; that is, the agent receives large total reward for small objective function value.

*Definition 6* (reward function). Let $K$ denote the number of decision-making epoch during an episode, $t_u$ $(0 \le u < K)$ the time at the $u$th decision-making epoch, $T_{i,u}$ $(1 \le i \le m, 1 \le u \le K)$ the job type of the lot which machine $i$ processes during time interval $(t_{u-1}, t_u]$, $T^0_{i,u}$ the job type of the lot which precedes the lot machine $i$ processes during time interval $(t_{u-1}, t_u]$, and $Y_j(t_u)$ the processed volume of job type $j$ by time $t_u$. It follows that

$$Y_j(t_u) - Y_j(t_{u-1}) = \sum_{i=1}^{m} \frac{(t_u - t_{u-1})\,\delta(i,j)\,L}{s_{T^0_{i,u},T_{i,u}} + p_{i,T_{i,u}}}, \quad (21)$$

where $\delta(i,j)$ is an indicator function defined as

$$\delta(i,j) = \begin{cases} 1, & T_{i,u} = j, \\ 0, & T_{i,u} \ne j. \end{cases} \quad (22)$$

Let $r_u$ $(u = 1, 2, \ldots, K)$ denote the reward function at the $u$th decision-making epoch. $r_u$ is defined as

$$r_u = \sum_{j=1}^{n} \min\left\{\sum_{i=1}^{m} \frac{(t_u - t_{u-1})\,\delta(i,j)\,L}{s_{T^0_{i,u},T_{i,u}} + p_{i,T_{i,u}}}, \left[D_j - Y_j(t_{u-1})\right]^+\right\} w_j + \max\left\{\sum_{i=1}^{m} \frac{(t_u - t_{u-1})\,\delta(i,j)\,L}{s_{T^0_{i,u},T_{i,u}} + p_{i,T_{i,u}}} - \left[D_j - Y_j(t_{u-1})\right]^+, 0\right\} \times \frac{w_j}{M}. \quad (23)$$

The reward function has the following property.

**Theorem 7.** *Maximization of the total reward $R$ in an episode is equivalent to minimization of objective function* (5).

*Proof.* The total reward in an episode is

$$\begin{aligned} R &= \sum_{u=1}^{K} r_u \\ &= \sum_{u=1}^{K}\sum_{j=1}^{n} \min\left\{\sum_{i=1}^{m} \frac{(t_u - t_{u-1})\,\delta(i,j)\,L}{s_{T^0_{i,u},T_{i,u}} + p_{i,T_{i,u}}}, \left[D_j - Y_j(t_{u-1})\right]^+\right\} w_j \\ &\quad + \max\left\{\sum_{i=1}^{m} \frac{(t_u - t_{u-1})\,\delta(i,j)\,L}{s_{T^0_{i,u},T_{i,u}} + p_{i,T_{i,u}}} - \left[D_j - Y_j(t_{u-1})\right]^+, 0\right\} \times \frac{w_j}{M} \\ &= \sum_{j=1}^{n}\sum_{u=1}^{K} \min\left\{\sum_{i=1}^{m} \frac{(t_u - t_{u-1})\,\delta(i,j)\,L}{s_{T^0_{i,u},T_{i,u}} + p_{i,T_{i,u}}}, \left[D_j - Y_j(t_{u-1})\right]^+\right\} \times w_j \\ &\quad + \max\left\{\sum_{i=1}^{m} \frac{(t_u - t_{u-1})\,\delta(i,j)\,L}{s_{T^0_{i,u},T_{i,u}} + p_{i,T_{i,u}}} - \left[D_j - Y_j(t_{u-1})\right]^+, 0\right\} \times \frac{w_j}{M}. \end{aligned} \quad (24)$$

It is easy to show that

$$Y_j = \sum_{u=1}^{K}\sum_{i=1}^{m} \frac{(t_u - t_{u-1})\,\delta(i,j)\,L}{s_{T^0_{i,u},T_{i,u}} + p_{i,T_{i,u}}}. \quad (25)$$

It follows that

$$\begin{aligned} R &= \sum_{j=1}^{n}\left[w_j \min\{D_j, Y_j\} + \frac{w_j}{M}\max\{0, Y_j - D_j\}\right] \\ &= \sum_{j\in\Omega_1}\left[w_j D_j + \frac{w_j}{M}(Y_j - D_j)\right] + \sum_{j\in\Omega_2} w_j Y_j \\ &= \sum_{j=1}^{n} w_j D_j - \left\{\sum_{j\in\Omega_1}\left[-\frac{w_j}{M}(Y_j - D_j)\right] + \sum_{j\in\Omega_2} w_j(D_j - Y_j)\right\} \\ &= \sum_{j=1}^{n} w_j D_j - \sum_{j=1}^{n}\left[w_j(D_j - Y_j)^+ - \frac{w_j}{M}(Y_j - D_j)^+\right], \end{aligned} \quad (26)$$

where $\Omega_1 = \{j \mid Y_j > D_j\}$ and $\Omega_2 = \{j \mid Y_j \le D_j\}$. Since $\sum_{j=1}^{n} w_j D_j$ is a constant, it follows that

$$\max R \Longleftrightarrow \min \sum_{j=1}^{n}\left[w_j(D_j - Y_j)^+ - \frac{w_j}{M}(Y_j - D_j)^+\right]. \quad (27)$$

□

## 3. The Reinforcement Learning Algorithm

The chip attach scheduling problem is converted into an RL problem with terminal state in Section 2. To apply $Q$-learning to solve this RL problem, another issue arises, that is, how to tailor $Q$-learning algorithm in this particular context. Since some state variables are continuous, the state space is infinite. This RL system is not in tabular form, and it is impossible to maintain $Q$-values for all state-action pairs. Thus, we use linear function with gradient-descent method to approximate the $Q$-value function. $Q$-values are represented as linear combination of a set of basis functions, $\Phi_k(s)$ $(1 \le k \le 4m+n)$, as shown in the next formula:

$$Q(s,a) = \sum_{k=1}^{4m+n} c_k^a \Phi_k(s), \tag{28}$$

where $c_k^a$ $(1 \le a \le 6, 1 \le k \le 4m+n)$ are the weights of basis functions. Each state variable corresponds to a basis function. The following basis functions are defined to normalize the state variables:

$$\Phi_k(s) = \begin{cases} \dfrac{T_k^0}{n} \quad (1 \le k \le m), \\ \dfrac{T_{k-m}}{n} \quad (m+1 \le k \le 2m), \\ \dfrac{t_{k-2m}}{\max\left\{s_{j1,j2} + p_{j2} \mid 1 \le j1 \le n, 1 \le j2 \le n\right\}} \\ \qquad\qquad (2m+1 \le k \le 3m), \\ \dfrac{d_{k-3m}}{D_{k-3m}} \quad (3m+1 \le k \le 3m+n), \\ \dfrac{e_{k-3m-n}}{\text{TH}} \quad (3m+n+1 \le k \le 4m+n). \end{cases} \tag{29}$$

Let $C^a$ denote the vector of weights of basis functions as follows:

$$C^a = \left(c_1^a, c_2^a, \ldots, c_{4m+n}^a\right)^T. \tag{30}$$

The RL algorithm is presented as Algorithm 8, where $\alpha$ is learning rate, $\gamma$ is a discount factor, $E(a)$ is the vector of eligibility traces for action $a$, $\delta(a)$ is an error variable for action $a$, and $\lambda$ is a factor for updating eligibility traces.

*Algorithm 8.* $Q$-learning with linear gradient-descent function approximation for chip attach scheduling.

```
Initialize $C^a$ and $E(a)$ randomly. Set parameters $\alpha$, $\gamma$, and $\lambda$.
Let num_episode denote the number of episodes having been run. Set num_episode = 0.
While num_episode < MAX_EPISODE do
Set the current decision-making state $s \leftarrow s_0$.
While at least one of state variables $e_i$ $(1 \le i \le m)$ is larger than zero do
    Select action $a$ for state $s$ by $\varepsilon$-greedy policy.
    Implement action a. Determine the next event for triggering state transition and the sojourn time. Once any machine completes processing a lot or any machine's normal production time is completely scheduled, the system transfers into a new decision-making state, $s'$ ($e_i'$ $(1 \le i \le m)$ is a component of $s'$).
    Compute reward $r_{s,s'}^a$.
    Update the vector of weights in the approximate $Q$-value function of action a:
```

$$\begin{aligned} \delta(a) &\longleftarrow r_{s,s'}^a + \gamma \max_{a'} Q\left(s', a'\right) - Q(s,a), \\ E(a) &\longleftarrow \lambda E(a) + \nabla_{C^a} Q(s,a), \\ C^a &\longleftarrow C^a + \alpha\delta(a) E(a). \end{aligned} \tag{31}$$

```
    Set $s \leftarrow s'$.
    If $e_i = 0$ holds for all $i$ $(1 \le i \le m)$, set num_episode = num_episode + 1.
```

## 4. Experiment Results

In the past, the company used a manual process to conduct chip attach scheduling. A heuristic algorithm called Largest Weight First (LWF) was used as follows.

*Algorithm 9* (Largest Weight First (LWF) heuristics). Initialize SM with the set of all machines (i.e., SM $= \{i \mid 1 \le i \le m\}$) and define SJ as Step 1 in Algorithm 2. Initialize $e_i$ $(1 \le i \le m)$ with each machine's normal production time. Set $Y_j = I_j$, where $I_j$ is the initial production volume of job type $j$.

*Step 1.* Schedule the job types in decreasing order of weights in order to meet their TPVs.

```
While SJ $\neq \Phi$ and SM $\neq \Phi$ do
    Choose job $q$ with $q = \text{argmax}\{w_j \mid j \in \text{SJ}\}$.
    While SM $\cap M_q \neq \Phi$ and $Y_q < D_q$ do
        Choose machine $i$ to process job $q$, with $i = \text{argmin}\{p_{k,q}/w_q \mid k \in \text{SM} \cap M_q\}$.
        If $e_i - s_{T_i^0,q} < (D_q - Y_q)p_{i,q}$, then
            set $Y_q \leftarrow Y_q + L(e_i - s_{T_i^0,q})/p_{i,q}$, $e_i = 0$, and remove $i$ from SM;
        else, set $e_i \leftarrow e_i - s_{T_i^0,q} - (D_q - Y_q)p_{i,q}/L$, and $Y_q = D_q$.
        $T_i^0 = q$
```

*Step 2.* Allocate the excess production capacity.

```
If SM $\neq \Phi$, then
    For each machine $i$ $(i \in \text{SM})$,
        Choose job $j$ with $j = \text{argmax}\{(e_i - s_{T_i^0,q})w_q/p_{i,q} \mid 1 \le q \le n\}$, set $Y_j \leftarrow Y_j + L(e_i - s_{T_i^0,j})/p_{i,j}$, $e_i = 0$.
```

Table 1: Comparison of objective function values using heuristics and $Q$-learning.

| Dataset no. | WSPT | MWSPT | RA | LFM-MWSPT | LFM-RA | LWF | $Q$-Learning |
|---|---|---|---|---|---|---|---|
| 1 | 88.867 | 78.116 | 59.758 | 87.689 | 57.582 | 42.253 | −3.8613 |
| 2 | 138.44 | 135.86 | 110.747 | 126.69 | 109.07 | 95.926 | 7.6657 |
| 3 | 119.01 | 108.75 | 124.09 | 104.25 | 121.90 | 83.332 | 23.775 |
| 4 | 83.681 | 60.797 | 39.073 | 69.405 | 45.575 | 33.920 | −4.4275 |
| 5 | 129.38 | 128.47 | 96.960 | 109.17 | 99.827 | 89.863 | 21.414 |
| 6 | 70.840 | 55.692 | 51.108 | 66.213 | 51.041 | 16.930 | −5.4467 |
| 7 | 120.90 | 100.60 | 95.399 | 109.33 | 90.754 | 76.422 | 27.374 |
| 8 | 102.42 | 107.80 | 116.56 | 103.33 | 107.62 | 93.663 | 11.840 |
| 9 | 94.606 | 87.914 | 81.763 | 88.812 | 80.331 | 60.164 | 33.036 |
| 10 | 90.803 | 88.164 | 90.773 | 87.926 | 88.56293 | 56.307 | 22.798 |
| 11 | 111.13 | 88.287 | 82.916 | 97.882 | 85.605 | 60.160 | 16.493 |
| 12 | 100.29 | 89.005 | 86.692 | 95.836 | 78.342 | 60.744 | −3.8617 |
| Average | 104.19 | 94.123 | 86.321 | 95.547 | 84.685 | 64.147 | 12.233 |

Table 2: Comparison of unsatisfied TPV index using heuristics and $Q$-learning.

| Dataset no. | WSPT | MWSPT | RA | LFM-MWSPT | LFM-RA | LWF | $Q$-learning |
|---|---|---|---|---|---|---|---|
| 1 | 0.1179 | 0.1025 | 0.0789 | 0.1170 | 0.0754 | 0.0554 | 0.0081 |
| 2 | 0.1651 | 0.1611 | 0.1497 | 0.1499 | 0.1482 | 0.1421 | 0.0137 |
| 3 | 0.1421 | 0.1289 | 0.1475 | 0.1227 | 0.1455 | 0.0987 | 0.0691 |
| 4 | 0.1540 | 0.1104 | 0.0716 | 0.1258 | 0.0854 | 0.0614 | 0.0088 |
| 5 | 0.1588 | 0.1564 | 0.1186 | 0.1303 | 0.1215 | 0.1094 | 0.0571 |
| 6 | 0.1053 | 0.0819 | 0.0757 | 0.1006 | 0.0762 | 0.0248 | 0.0137 |
| 7 | 0.1462 | 0.1209 | 0.1150 | 0.1292 | 0.1082 | 0.0917 | 0.0582 |
| 8 | 0.1266 | 0.1324 | 0.1437 | 0.1272 | 0.1309 | 0.1150 | 0.0381 |
| 9 | 0.1315 | 0.1211 | 0.1133 | 0.1249 | 0.1127 | 0.0828 | 0.0815 |
| 10 | 0.1154 | 0.1112 | 0.1151 | 0.1105 | 0.1110 | 0.0709 | 0.0536 |
| 11 | 0.1544 | 0.1215 | 0.1146 | 0.1387 | 0.1204 | 0.0827 | 0.0690 |
| 12 | 0.1262 | 0.1112 | 0.1088 | 0.1194 | 0.1002 | 0.0758 | 0.0118 |
| Average | 0.1370 | 0.1216 | 0.1112 | 0.1247 | 0.1113 | 0.0842 | 0.0402 |

The chip attach station consists of 10 machines and normally processes more than ten job types. We selected 12 sets of industrial data for experiments comparing the $Q$-learning algorithm (Algorithm 8) and the six heuristics (Algorithms 1–5, 9): WSPT, MWSPT, RA, LFM-MWSPT, LFM-RA, and LWF. For each dataset, $Q$-learning repeatedly solves the scheduling problem 1000 times and selects the optimal schedule of the 1000 solutions. Table 1 shows the objective function values of all datasets using the seven algorithms. Individually, any of WSPT, MWSPT, RA, LFM-MWSPT, and LFM-RA obtains larger objective function values than LWF for every dataset. Nevertheless, taking WSPT, MWSPT, RA, LFM-MWSPT, and LFM-RA as actions, $Q$-learning algorithm achieves an objective function value much smaller than LWF for each dataset. In Tables 1–4, the bottom row presents the average value over all datasets. As shown in Table 1, the average objective function value of $Q$-learning is only 12.233, less than that of LWF, 66.147, by a large amount of 80.92%.

Besides objective function value, we propose two indices, unsatisfied TPV index and unsatisfied job type index, to measure the performance of the seven algorithms. Unsatisfied TPV index (UPI) is defined as formula (32) and indicates the weighted proportion of unfinished Target Production Volume. Table 2 compares UPIs of all datasets using seven algorithms. Also, any of WSPT, MWSPT, RA, LFM-MWSPT, and LFM-RA individually obtains larger UPI than LWF for each dataset. However, $Q$-learning algorithm achieves smaller UPI than LWF does for each dataset. The average UPI of $Q$-learning is only 0.0402, less than that of LWF, 0.0842, by a large amount of 52.20%. Let $J$ denote the set $\{j \mid 1 \le j \le n, Y_j < D_j\}$. Unsatisfied job type index (UJTI) is defined as formula (33) and indicates the weighted proportion of the job types whose TPVs are not completely satisfied. Table 3 compares UJTIs of all datasets using seven algorithms. With most datasets, $Q$-learning algorithm achieves smaller UJTIs than LWF. The average UJTI of $Q$-learning is 0.0802, which is remarkably less than that of LWF, 0.1176, by 31.81%. Consider

$$\mathrm{UPI} = \frac{\sum_{j=1}^{n} w_j \left(D_j - Y_j\right)^+}{\sum_{j=1}^{n} w_j D_j}, \qquad (32)$$

$$\mathrm{UJTI} = \frac{\sum_{j=J} w_j}{\sum_{j=1}^{n} w_j}. \qquad (33)$$

Table 3: Comparison of unsatisfied job type index using heuristics and $Q$-learning.

| Dataset no. | WSPT | MWSPT | RA | LFM-MWSPT | LFM-RA | LWF | $Q$-learning |
|---|---|---|---|---|---|---|---|
| 1 | 0.1290 | 0.2615 | 0.1667 | 0.1650 | 0.1793 | 0.0921 | 0.0678 |
| 2 | 0.1924 | 0.2953 | 0.2302 | 0.2650 | 0.2097 | 0.1320 | 0.0278 |
| 3 | 0.2250 | 0.3287 | 0.2126 | 0.2564 | 0.2278 | 0.0921 | 0.0921 |
| 4 | 0.0781 | 0.2987 | 0.0278 | 0.1290 | 0.0828 | 0.0278 | 0.0278 |
| 5 | 0.2924 | 0.3169 | 0.3002 | 0.2224 | 0.2632 | 0.2055 | 0.0571 |
| 6 | 0.2290 | 0.3062 | 0.1817 | 0.2290 | 0.1632 | 0.0571 | 0.0278 |
| 7 | 0.2650 | 0.2987 | 0.2160 | 0.2529 | 0.2075 | 0.1320 | 0.1647 |
| 8 | 0.1924 | 0.3225 | 0.2067 | 0.1813 | 0.1696 | 0.1320 | 0.1320 |
| 9 | 0.2221 | 0.3250 | 0.1403 | 0.1892 | 0.2073 | 0.1320 | 0.0749 |
| 10 | 0.2621 | 0.3304 | 0.2667 | 0.2859 | 0.2708 | 0.1781 | 0.0678 |
| 11 | 0.2029 | 0.2896 | 0.2578 | 0.2194 | 0.2220 | 0.1381 | 0.1542 |
| 12 | 0.1924 | 0.3271 | 0.2302 | 0.1838 | 0.2182 | 0.0921 | 0.0678 |
| Average | 0.2069 | 0.3084 | 0.2031 | 0.2149 | 0.2018 | 0.1176 | 0.0802 |

Table 4: Comparison of the total setup time using heuristics and $Q$-learning.

| Dataset no. | WSPT | MWSPT | RA | LFM-MWSPT | LFM-RA | LWF | $Q$-learning |
|---|---|---|---|---|---|---|---|
| 1 | 0.8133 | 0.8497 | 0.3283 | 0.7231 | 0.3884 | 0.3895 | 1.0000 |
| 2 | 0.8333 | 1.3000 | 0.4358 | 0.7564 | 0.5263 | 0.4094 | 1.0000 |
| 3 | 0.8712 | 1.1633 | 0.4207 | 0.6361 | 0.4937 | 0.4298 | 1.0000 |
| 4 | 1.1280 | 0.7123 | 0.4629 | 0.8516 | 0.5139 | 0.4318 | 1.0000 |
| 5 | 0.9629 | 1.3121 | 0.4179 | 0.8597 | 0.5115 | 0.3873 | 1.0000 |
| 6 | 0.7489 | 1.0393 | 0.4104 | 0.7489 | 0.4542 | 0.4074 | 1.0000 |
| 7 | 1.7868 | 2.2182 | 0.8223 | 1.4069 | 1.0125 | 0.4174 | 1.0000 |
| 8 | 0.6456 | 0.8508 | 0.4055 | 0.6694 | 0.5053 | 0.3795 | 1.0000 |
| 9 | 0.9245 | 0.9946 | 0.5013 | 0.7821 | 0.6694 | 0.4163 | 1.0000 |
| 10 | 1.1025 | 1.7875 | 0.6703 | 1.0371 | 0.9079 | 0.4894 | 1.0000 |
| 11 | 0.9973 | 1.3655 | 0.3994 | 0.9686 | 0.5129 | 0.4066 | 1.0000 |
| 12 | 0.7904 | 1.1111 | 0.4419 | 0.6195 | 0.5081 | 0.4258 | 1.0000 |
| Average | 0.9671 | 1.2254 | 0.4764 | 0.8383 | 0.5837 | 0.4158 | 1.0000 |

Table 4 shows the total setup time of all datasets using seven algorithms. For the reason of commercial confidentiality, we used the normalized data with the setup time of a dataset divided by the result of this dataset using $Q$-learning. Thus, the total setup times of all datasets by $Q$-learning are converted into one and the data of the six heuristics are adjusted accordingly. $Q$-learning algorithm requires more than twice of setup time than LWF does for each dataset. The average accumulated setup time of LWF is only 41.58 percents of that of $Q$-learning.

The previous experimental results reveal that for the whole scheduling tasks, any individual one of the five action heuristics (WSPT, MWSPT, RA, LFM-MWSPT, and LFM-RA) for $Q$-learning performs worse than LWF heuristics. However, $Q$-learning greatly outperforms LWF in terms of the three performance measures, the objective function value, UPI, and UJTI. This demonstrates that some action heuristics provide better actions than LWF heuristics at some states. During repeatedly solving the scheduling problem, $Q$-learning system perceives the insights of the scheduling problem automatically and adjusts its actions towards the optimal ones facing different system states. The actions at all states form a new optimized policy which is different from any policies following any individual action heuristics or LWF heuristics. That is, $Q$-learning incorporates the merit of five alternative heuristics, uses them to schedule jobs flexibly, and obtains results much better than any individual action heuristics and LWF heuristics. In the experiments, $Q$-learning achieves high-quality schedules at the cost of inducing more setup time. In other words, $Q$-learning utilizes the machines more efficiently by increasing conversions among a variety of job types.

## 5. Conclusions

We apply $Q$-learning to study lot-based chip attach scheduling in back-end semiconductor manufacturing. To apply reinforcement learning to scheduling, the critical issue being conversion of scheduling problems into RL problems. We convert chip attach scheduling problem into a particular SMDP problem by Markovian state representation. Five heuristic algorithms, WSPT, MWSPT, RA, LFM-MWSPT, and LFM-RA, are selected as actions so as to utilize prior domain knowledge. Reward function is directly related to scheduling objective function, and we prove that maximizing the accumulated reward is equivalent to minimizing the objective function. Gradient-descent linear function approximation is combined with $Q$-learning algorithm.

$Q$-learning exploits the insight structure of the scheduling problem by solving it repeatedly. It learns a domain-specific policy from the experienced episodes through interaction and then applies it to latter episodes. We define two indices, unsatisfied TPV index and unsatisfied job type index, together with objective function value to measure the performance of $Q$-learning and the heuristics. Experiments with industrial datasets show that $Q$-learning apparently outperforms six heuristic algorithms: WSPT, MWSPT, RA, LFM-MWSPT, LFM-RA, and LWF. Compared with LWF, $Q$-learning achieves reduction of the three performance measures, respectively, by an average level of 52.20%, 31.81%, and 80.92%. With $Q$-learning, chip attach scheduling is optimized through increasing effective job type conversions.

## Disclosure

Given the sensitive and proprietary nature of the semiconductor manufacturing environment, we use normalized data in this paper.

## Acknowledgments

This project is supported by the National Natural Science Foundation of China (Grant no. 71201026), Science and Technological Program for Dongguan's Higher Education, Science and Research, and Health Care Institutions (no. 2011108102017), and Humanities and Social Sciences Program of Ministry of Education of China (no. 10YJC630405).

## References

[1] M. X. Weng, J. Lu, and H. Ren, "Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective," *International Journal of Production Economics*, vol. 70, no. 3, pp. 215–226, 2001.

[2] M. Gairing, B. Monien, and A. Woclaw, "A faster combinatorial approximation algorithm for scheduling unrelated parallel machines," in *Automata, Languages and Programming*, vol. 3580 of *Lecture Notes in Computer Science*, pp. 828–839, 2005.

[3] G. Mosheiov, "Parallel machine scheduling with a learning effect," *Journal of the Operational Research Society*, vol. 52, no. 10, pp. 1–5, 2001.

[4] G. Mosheiov and J. B. Sidney, "Scheduling with general job-dependent learning curves," *European Journal of Operational Research*, vol. 147, no. 3, pp. 665–670, 2003.

[5] L. Yu, H. M. Shih, M. Pfund, W. M. Carlyle, and J. W. Fowler, "Scheduling of unrelated parallel machines: an application to PWB manufacturing," *IIE Transactions*, vol. 34, no. 11, pp. 921–931, 2002.

[6] K. R. Baker and J. W. M. Bertrand, "A dynamic priority rule for scheduling against due-dates," *Journal of Operations Management*, vol. 3, no. 1, pp. 37–42, 1982.

[7] J. J. Kanet and X. Li, "A weighted modified due date rule for sequencing to minimize weighted tardiness," *Journal of Scheduling*, vol. 7, no. 4, pp. 261–276, 2004.

[8] R. V. Rachamadugu and T. E. Morton, "Myopic heuristics for the single machine weighted tardiness problem," Working Paper 28-81-82, Graduate School of Industrial Administration, Garnegie-Mellon University, 1981.

[9] A. Volgenant and E. Teerhuis, "Improved heuristics for the n-job single-machine weighted tardiness problem," *Computers and Operations Research*, vol. 26, no. 1, pp. 35–44, 1999.

[10] D. C. Carroll, *Heuristic sequencing of jobs with single and multiple components [Ph.D. thesis]*, Sloan School of Management, MIT, 1965.

[11] A. P. J. Vepsalainen and T. E. Morton, "Priority rules for job shops with weighted tardiness costs," *Management Science*, vol. 33, no. 8, pp. 1035–1047, 1987.

[12] R. S. Russell, E. M. Dar-El, and B. W. Taylor, "A comparative analysis of the COVERT job sequencing rule using various shop performance measures," *International Journal of Production Research*, vol. 25, no. 10, pp. 1523–1540, 1987.

[13] J. Bank and F. Werner, "Heuristic algorithms for unrelated parallel machine scheduling with a common due date, release dates, and linear earliness and tardiness penalties," *Mathematical and Computer Modelling*, vol. 33, no. 4-5, pp. 363–383, 2001.

[14] C. F. Liaw, Y. K. Lin, C. Y. Cheng, and M. Chen, "Scheduling unrelated parallel machines to minimize total weighted tardiness," *Computers and Operations Research*, vol. 30, no. 12, pp. 1777–1789, 2003.

[15] D. W. Kim, D. G. Na, and F. F. Chen, "Unrelated parallel machine scheduling with setup times and a total weighted tardiness objective," *Robotics and Computer-Integrated Manufacturing*, vol. 19, no. 1-2, pp. 173–181, 2003.

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

[17] C. J. C. H. Watkins, *Learning from delayed rewards [Ph.D. thesis]*, Cambridge University, 1989.

[18] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[19] T. Jaakkola, M. I. Jordan, and S. P. Singh, "On the convergence of stochastic iterative dynamic programming algorithms," *Neural Computation*, vol. 6, pp. 1185–1201, 1994.

[20] J. N. Tsitsiklis, "Asynchronous stochastic approximation and Q-learning," *Machine Learning*, vol. 16, no. 3, pp. 185–202, 1994.

[21] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*, Athena Scientific, Belmont, Mass, USA, 1996.

[22] S. Riedmiller and M. Riedmiller, "A neural reinforcement learning approach to learn local dispatching policies in production scheduling," in *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, 1999.

[23] M. E. Aydin and E. Öztemel, "Dynamic job-shop scheduling using reinforcement learning agents," *Robotics and Autonomous Systems*, vol. 33, no. 2, pp. 169–178, 2000.

[24] J. Hong and V. V. Prabhu, "Distributed reinforcement learning control for batch sequencing and sizing in just-in-time manufacturing systems," *Applied Intelligence*, vol. 20, no. 1, pp. 71–87, 2004.

[25] Y. C. Wang and J. M. Usher, "Application of reinforcement learning for agent-based production scheduling," *Engineering Applications of Artificial Intelligence*, vol. 18, no. 1, pp. 73–82, 2005.

[26] B. C. Csáji, L. Monostori, and B. Kádár, "Reinforcement learning in a distributed market-based production control system," *Advanced Engineering Informatics*, vol. 20, no. 3, pp. 279–288, 2006.

[27] S. S. Singh, V. B. Tadić, and A. Doucet, "A policy gradient method for semi-Markov decision processes with application to call admission control," *European Journal of Operational Research*, vol. 178, no. 3, pp. 808–818, 2007.

[28] M. Kaya and R. Alhajj, "A novel approach to multiagent reinforcement learning: utilizing OLAP mining in the learning process," *IEEE Transactions on Systems, Man and Cybernetics Part C*, vol. 35, no. 4, pp. 582–590, 2005.

[29] C. D. Paternina-Arboleda and T. K. Das, "A multi-agent reinforcement learning approach to obtaining dynamic control policies for stochastic lot scheduling problem," *Simulation Modelling Practice and Theory*, vol. 13, no. 5, pp. 389–406, 2005.

[30] C. E. Mariano-Romero, V. H. Alcocer-Yamanaka, and E. F. Morales, "Multi-objective optimization of water-using systems," *European Journal of Operational Research*, vol. 181, no. 3, pp. 1691–1707, 2007.

[31] D. Vengerov, "A reinforcement learning framework for utility-based scheduling in resource-constrained systems," *Future Generation Computer Systems*, vol. 25, no. 7, pp. 728–736, 2009.

[32] K. Iwamura, N. Mayumi, Y. Tanimizu, and N. Sugimura, "A study on real-time scheduling for holonic manufacturing systems—determination of utility values based on multi-agent reinforcement learning," in *Proceedings of the 4th International Conference on Industrial Applications of Holonic and Multi-Agent Systems*, pp. 135–144, Linz, Austria, 2009.

[33] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, Prentice Hall, Englewoods Cliffs, NJ, USA, 2nd edition, 2002.