A Reinforcement Learning Approach to Dynamic Scheduling in a Product-Mix Flexibility Environment

YEOU-REN SHIUE1, KEN-CHUAN LEE2, AND CHAO-TON SU2

1Department of Management Engineering, Fujian Business University, Fuzhou 350012, China
2Department of Industrial Engineering and Engineering Management, National Tsing Hua University, Hsinchu 30013, Taiwan

ABSTRACT Machine bottlenecks, resulting from shifting and unbalanced machine loads caused by resource capacity limitations, impair product-mix flexibility production systems. Thus, the knowledge base (KB) of a dynamic scheduling control system should be dynamic and include a knowledge revision mechanism for monitoring crucial changes that occur in the production system. In this paper, reinforcement learning (RL)-based dynamic scheduling and a selection mechanism for multiple dynamic scheduling rules (MDSRs) are proposed to support the operating characteristics of a flexible manufacturing system (FMS) and semiconductor wafer fabrication (FAB). The proposed RL-based dynamic scheduling MDSR selection mechanism consisted of initial MDSR KB generation and revision phases. According to various performance criteria, the presented approach yields a system performance that is superior to those of the fixed-decision scheduling approach, the machine learning classification approach, and the classical MDSR selection mechanism.

INDEX TERMS Manufacturing execution system, dynamic scheduling, machine learning, reinforcement learning, Q-learning.

I. INTRODUCTION

In the 21st century, the advanced high-tech industry will inevitably face global market competition. This highly dynamic and customer-demand-oriented production system is characterized by both rapid changes in product-mix ratio and the ability to deliver quickly. A possible solution to this problem is the implementation of an efficient scheduling mechanism to increase product throughput and shorten the mean production cycle time (or reduce the mean flow time) [1], [2]. This product-mix flexibility scheduling mechanism can improve customer service and enhance market competitiveness. Many flexible job shop scheduling methods based on swarm intelligence or evolutionary algorithms have been introduced by [3]–[5]. However, these scheduling methods cannot handle the problem that new customer orders continue to enter the system and the machine bottleneck shifting problem. In addition, these approaches can be very time consuming and difficult to implement in a real-world production system [6]. Therefore, many researchers [7]–[13] believe that the development of an effective dynamic scheduling mechanism is crucial for satisfying the various performance criteria of a manufacturing execution system (MES).

In a dynamic scheduling mechanism, various scheduling rules are applied in a dynamic and multi-pass manner to select the nearest-to-optimal dispatching (i.e., scheduling) strategy among the feasible alternatives at each scheduling decision point and thereby satisfy shop floor performance criteria [14], [15]. Previous studies [16], [17] have indicated that dynamic scheduling includes two main approaches: the multi-pass simulation technique [18], [19] and the machine learning technique [20]–[23]. Multi-pass simulations examine candidate scheduling rules and select the best strategy according to simulation information, such as the current state of the system and performance criteria for each scheduling interval. However, the multi-pass simulation technique is unsuitable for dynamic scheduling as it requires exhaustive computing power to choose the best scheduling rule for each scheduling decision point.

In the machine learning technique for dynamic scheduling, a training dataset is generated that uses system simulations to determine the best scheduling rule for each possible
system status. However, generating training datasets and learning processes to obtain a dynamic scheduling knowledge base (KB) is often time consuming. A dynamic scheduling KB can provide fast and acceptable solutions, allowing the shop floor to make immediate decisions and the system to meet the operational characteristics required of a dynamic production system [16]. Three major machine learning classification methods for establishing a dynamic scheduling KB were used in earlier research [22]: neural networks [24], C4.5 decision trees [25], and support vector machines [26]. Many studies [27]–[30] apply the above machine learning classification methods to develop dynamic scheduling in flexible manufacturing systems (FMSs) and wafer fabrication (FAB) environments.

According to previous research [20], there are two ways to determine the scheduling decision rules in dynamic scheduling research: a single dynamic scheduling rule (SDSR) and multiple dynamic scheduling rules (MDSRs) for a production system. The SDSR typically assigns an individual heuristic scheduling rule to all machines in a system during a given time window (i.e., the scheduling period) using machine learning classification methods, whereas in the MDSR approach, different scheduling rules are assigned to all machines in a system. Figure 1 depicts the role of the MDSR mechanism in FAB. For FAB input control (i.e., order release control) and the selection of wafer lots through the intrabay and stocker, the MDSR mechanism selects the decision rules pertaining to (1) the order release workload level (WL), (2) the wafer lot with the shortest processing time (SPT), and (3) the wafer lot with the earliest due date (EDD) as the scheduling decision variables for wafer lot selection in the next scheduling period. Ishii and Talavage [31] presented a heuristic algorithm that applies the MDSR strategy to bottlenecks and unbalanced machines by using predictions based on multi-pass simulations, which confirmed that the MDSR strategy can improve the performance of a FMS by up to 15.9% compared with the best result obtained using the SDSR strategy. However, their method did not work well with dynamic scheduling using machine learning classification methods.

In a MDSR-based dynamic scheduling KB using the machine learning classification method, the main drawback is that the class label (the scheduling decision rule) for assigning training samples must be specified in advance. For instance, for a given set of system features, the best scheduling decision rule can be determined after a simulation is performed for each candidate scheduling rule. The resulting MDSRs are considered training sample class labels. However, this training process becomes considerably time-consuming because scheduling decision rules must be determined for each scheduling period [32]. Furthermore, machine learning classification approaches, such neural networks and C4.5 decision trees, do not achieve satisfactory global performance for the production system. Thus, although the best decision rule can be determined for each scheduling decision variable, the combination of all of the scheduling decision rules may not simultaneously satisfy the global objective function.

Previous studies [28], [29], [33], [34] have indicated that a FAB (or FMS) provides a made-to-order production environment in which bottlenecks often shift with unbalanced machine loads caused by complex product-mix orders, resulting in a rapid increase in the mean production cycle time. A dynamic scheduling KB should be dynamic in a product-mix flexibility environment. Therefore, an approach that automatically revises the scheduling KB if critical changes occur in the production system should be developed. This approach can solve the problems of machine bottleneck shifting and unbalanced machine loads caused by resource capacity limitations. Some studies [11], [16], [30], [35] have indicated that one of the restrictions of dynamic scheduling using machine learning classification approaches is the lack of a revised mechanism in the dynamic scheduling KB. Recent advances in information and communication technology (ICT) using reinforcement learning (RL) [36] has been widely used [37] in robot motion control [38], [39], industrial process control [40], [41], and production scheduling [15], [42]–[46], and internet of things (IoT) [47]. ICT learns the optimal policy through interaction with the environment without the model of the environment.

Based on the above discussion, an effective MDSR-based dynamic scheduling system should include a KB revision mechanism. The RL approach, which interacts with the product-mix flexibility production environment, responds to rewards or penalties based on the results of each action (i.e., MDSR) and can learn to select the most suitable MDSR to achieve production performance goals. Q-learning [48], which is an algorithm of the RL family, is used for most of the aforementioned studies on production scheduling problems. The goal of Q-learning is to learn a policy that tells an agent which finite action to take based on a set of discrete circumstances. Under this scheme, agents have the ability to learn to act optimally in Markovian domains by experiencing the consequences of actions without being required to build domain maps. The characteristics of the Q-learning algorithm described above render it suitable for processing
discrete-event-based dynamic scheduling problems. Wang and Usher [49] showed that this algorithm provides satisfactory results for a single machine dispatch rule selection problem when a Q-learning agent is used to select dispatch rules based on various production criteria. In recent years, some studies [42], [45] have been conducted using RL for dynamic order dispatching in the semiconductor industry. However, they used only the SDSR-based mechanism for an FAB. SDSR has its limitations in controlling large and complex systems, as mentioned above. Their research could incentivize future applications of RL-based MDSR approaches to dynamic scheduling systems experiencing complex product-mix order problems. Based on the aforementioned studies, to develop a MDSR-based mechanism for the dynamic scheduling system, dynamic scheduling should be able to maintain and update KB through RL during operations in response to changes in the system operating conditions. Therefore, the use of a RL agent to revise a dynamic scheduling KB should be evaluated.

In this research, we developed a dynamic scheduling system that uses a RL-based MDSR selection mechanism to support the complex product-mix flexibility environment. Based on this design philosophy, we present a RL-based MDSR selection mechanism by assigning various scheduling rules for all machines in a dynamically complex production system, such as a FMS or FAB, during a given scheduling period. Furthermore, the presented RL-based dynamic scheduling system can be employed to consider both the order release and dispatch rule, such as those of a FAB system. According to various production performance criteria, we believe that the presented methodology provides better production performance than those of a fixed-decision scheduling (dispatching) approach and the classical MDSR selection mechanism for dynamic scheduling.

The remainder of this paper is organized as follows: in section 2, we define the dynamic scheduling issue using the MDSR mechanism and solution methodology. Section 3 describes the initial MDSR KB generation phase, which is the first phase of the RL-based MDSR selection mechanism and supports the complex product-mix flexibility environment. Section 4 describes the MDSR KB revision phase, which is the second phase of the RL-based MDSR selection mechanism. In section 5, we show the experimental results of FMS and FAB case studies and analyze the proposed Q-learning-based agent approach as well as other approaches. Finally, in section 6, we provide the conclusions of this research and suggestions for future studies.

II. PROBLEM FORMULATION AND SOLUTION METHODOLOGY

A. DYNAMIC SCHEDULING SYSTEM EMPLOYING THE MDSR MECHANISM

The state of a production environment dynamically changes, and earlier research [18], [50] has indicated that the production performance can be improved by executing a multi-pass (i.e., dynamic) scheduling strategy according to the system status at each decision point over a series of short-term scheduling periods rather than using an individual heuristic scheduling rule over the complete entire scheduling planning period. Dynamic scheduling based on machine learning can quickly produce satisfactory solutions and meet the operational properties of production systems.

The dynamic scheduling system that employs the MDSR mechanism, denoted MDSR, can be presented by the 4-tuple \( \{O, D, L, S\} \), where \( S \) is the set of scheduling decision points \( \{s_1, s_2, \ldots, s_T\} \) denoted by a union of system features \( f \). At \( s_t \), the MDSR \( d_{next} \) for the next scheduling period is employed by the machine learning method \( l \in L \) based on system features \( f \) and the specific performance criterion \( o \in O \). Three types of performance criteria are typically studied in dynamic scheduling problems: throughput \( (T_P) \), mean cycle time \( (M_{CT}) \), and number of tardy \( (N_T) \) jobs (lots). In vector \( d_{next} = [d_1, d_2, \ldots, d_M]^T \), for \( d_m \in D \) and \( m = 1, 2, \ldots, M \), \( D \) is the candidate scheduling rule used in dynamic scheduling, and \( m \) is the number of machine cells in the production system. Therefore, the production performance measure for criterion \( o \) denoted \( \phi_{PM} \) for a long period of time in the dynamic scheduling of production systems employing the MDSR mechanism can be described as follows:

\[
\phi_{PM} = \sum_{t=1}^{T} \max_{s_t} MDSR(s_t, l), \quad o \in \{T_P\}
\]

\[
\phi_{PM} = \sum_{t=1}^{T} \min_{s_t} MDSR(s_t, l), \quad o \in \{M_{CT}, N_T\} \quad (1)
\]

Based on equation (1), the dynamic scheduling approach using the MDSR mechanism is defined by the system features and the specific machine learning approach for a short-term scheduling period under various performance criteria. In the long run, compared to dynamic scheduling with fixed-decision scheduling rules for each decision variable at the beginning of each production period under various performance criteria, production performance can be significantly improved using dynamic scheduling with the MDSR mechanism.

B. CASE STUDY

To verify the presented methodology, the proposed dynamic scheduling system employed two types of production systems: FMS and FAB. The proposed approach plays various roles in these two case studies. In the FMS case study, the dynamic scheduling system selects dispatch rules for each machine cell. In the FAB system, the dynamic scheduling system concurrently selects order release control and scheduling rules to meet the operational characteristics of the FAB system. Remaining deadlock-free in an important issue for the vehicle management of production systems [51]–[53]. Deadlocks are caused by concurrent stream contention or limited sharing of resources in the system. Therefore, when designing the physical specifications of the production system, how to avoid deadlock must be
considered. The following description illustrates the basic specifications of the two types of production systems:

1) **FMS Case Study**: This research case involves modification of the model adopted by Montazeri and Van Wassenhove [54]. The FMS model involves the following: (i) three types of machine cells, (ii) three load/unload stations, (iii) three automated guided vehicles, and (iv) input and central work-in-process buffers with enough production capacity to avoid system deadlock. The first two machine cells each have two machines, and the third machine cell has a single machine. This FMS model produces 11 types of parts, and the production routing and processing times are identical to those used in an earlier study [54].

2) **FAB Case Study**: The FAB system model used in this study is a simplified version of an International SEMATECH wafer FAB model [55]. This FAB system contains 12 failure-prone processing stations (with one or more identical but independent machines with a bay), 12 storage buffers in the stocker with enough production capacity (the maximum number in any stocker is rarely more than 150 FOUPs [55]) to avoid system deadlock, and one automated material handling system. Several major operation-related assumptions are based on earlier studies [56], [57]. Three types of processing technologies are available in this FAB model, and their processing times are identical to those used in previous research [57], [58].

### C. TRAINING DATA SET SPECIFICATION

A training dataset provides information on system patterns for learning the concepts describing each class. Hence, to build a MDSR KB, training examples need enough information to show this characteristic. In a MDSR KB, X denotes the training dataset generated using the training sample generation mechanism. A training example x ∈ X is expressed as \( f, d^{\text{next}} \), where \( d^{\text{next}} \) indicates the effective MDSRs for the subsequent scheduling period under performance criterion \( o \).

Table 1 lists 30 candidate system features. The principles for selecting these system features are according to those applied in earlier studies [22], [59]–[61], which employed machine learning to develop dynamic scheduling KBs. The need for dynamic dispatching rules arises from the fact that no particular dispatching rule has been proven to be optimal under a variety of shop configurations and operating conditions. Consequently, excessive efforts are not required for studying the best dispatching heuristics in various environments. Table 2 summarizes five dispatching rules applied for FMS control that were found to be effective in earlier studies [20], [62]. These dispatch rules can also be applied to a FAB intrabay station and used as stocker control decision rules. Table 3 summarizes five input control decision rules for FAB order release control.

### D. AN RL ALGORITHM

RL addresses the question of how an autonomous agent that senses and acts in its environment can learn to choose optimal actions to achieve its goal. In the RL framework, a learning agent must be able to perceive information in its environment; perceived information is identified as the current state of the environment. Then, the agent chooses an action to perform based on the perceived state. These interactions between the learning agent and its environment continue until the agent learns a decision-making policy that maximizes the total reward.

The original Q-learning algorithm has been presented by Watkins and Dayan [48], and the goal of this algorithm is to learn the state-action pair value, \( Q(s, a) \), which represents the long-term expected reward for each state-action pair (expressed by \( s \) and \( a \)). It has been proven that the \( Q \) values learned by applying this algorithm converge to the optimal

---

**TABLE 1. System features.**

| System Features | Description |
|-----------------|-------------|
| NJ              | Number of parts (wafer lot) in the FMS (FAB) |
| MUI             | Mean utilization of machines |
| SDU             | Standard deviation of machine utilization |
| MUL             | Mean utilization of load/unload stations |
| MDB             | Mean utilization of pallet buffer (stockers) |
| MUA             | Mean utilization of AGVs |
| MIP             | Maximum imminent processing time of candidate parts (wafer lot) in the FMS (FAB) |
| MIO             | Mean imminent processing time of candidate parts (wafer lot) in the FMS (FAB) |
| SDT             | Standard deviation of the imminent processing time of candidate parts (wafer lot) in the FMS (FAB) |
| MTP             | Minimum total processing time of candidate parts (wafer lot) in the FMS (FAB) |
| MUP             | Maximum mean total processing time of candidate parts (wafer lot) in the FMS (FAB) |
| MPT             | Mean total processing time of candidate parts (wafer lot) in the FMS (FAB) |
| SDPT            | Standard deviation of the total processing time of candidate parts (wafer lot) in the FMS (FAB) |
| MBT             | Minimum remaining processing time of candidate parts (wafer lot) in the FMS (FAB) |
| MRT             | Mean remaining processing time of candidate parts (wafer lot) in the FMS (FAB) |
| SDRT            | Standard deviation of the remaining processing time of candidate parts (wafer lot) in the FMS (FAB) |
| MST             | Minimum slack time of candidate parts (wafer lot) in the FMS (FAB) |
| MSST            | Mean slack time of candidate parts (wafer lot) in the FMS (FAB) |
| SDST            | Standard deviation of the slack time of candidate parts (wafer lot) in the FMS (FAB) |
| MCT             | Maximum tardiness of candidate parts (wafer lot) in the FMS (FAB) |
| MCTA            | Mean tardiness of candidate parts (wafer lot) in the FMS (FAB) |
| SDA             | Standard deviation of the tardiness of candidate parts (wafer lot) in the FMS (FAB) |
| MWL             | Maximum workload in front of any machine station in the FMS (FAB) |
| MWL             | Total workload in front of any machine station in the FMS (FAB) |
| MSST            | Mean stay time of candidate parts (wafer lot) in the FMS (FAB) |
| SPT             | Standard deviation of the stay time of candidate parts (wafer lot) in the FMS (FAB) |
| MCTD            | Mean time until due date of candidate parts (wafer lot) in the FMS (FAB) |
| SNID            | Standard deviation of the time from now until the due date of candidate parts (wafer lot) in the FMS (FAB) |

**TABLE 2. Dispatching rule.**

| Dispatching Rule | Description |
|------------------|-------------|
| DS               | Choose the parts (wafer lot) with minimum slack time |
| DED              | Choose the parts (wafer lot) with the earliest due date |
| SID              | Choose the parts (wafer lot) with the shortest imminent operation time |
| SPT              | Choose the parts (wafer lot) with the shortest processing time |
| SRPT             | Choose the parts (wafer lot) with the shortest remaining processing time |

**TABLE 3. Input (order release) control decision rules.**

| Decision Rule | Description |
|---------------|-------------|
| WLI            | Increase WL by 1% |
| WLI            | Increase WL by 5% |
| WLI            | Same WL |
| WD              | Decrease WL by 5% |
| WDI            | Decrease WL by 10% |

WL (workload level); for the sum of the remaining processing times at critical station 6 for all wafer lots is the FAB.
state-action values $Q$. The Q value of each action $(a)$ when performed in a state $(s)$ can be calculated as follows [48]:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[ r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right] \quad (2)$$

Each action consists of steps that represent a learning cycle, where $\alpha$ is the step size parameter influencing the learning rate, and $\gamma$ is the discount-rate parameter; $0 \leq \gamma \leq 1$, which affects the present value of future rewards. The Q-learning agent chooses an action to be implemented in the current state $s$ and recognizes the next state $s'$. Assuming that the agent implements the best strategy from state $s$, it will obtain a reward and update the $Q(s, a)$ value. In addition, the Q-learning agent can employ an exploration/exploitation strategy to maximize payoff.

E. SOLUTION METHODOLOGY

In this research, the suggested RL-based dynamic scheduling system applying the MDSR mechanism (displayed in Figure 2) includes an initial MDSR KB generation phase and a MDSR KB revision phase. First, a simulation model is run to produce a training dataset in the initial MDSR KB generation phase, and the system status number is determined by the two-level self-organizing map (SOM) algorithm [63]. The system status number is assigned to the initial Q-table and used to initialize the MDSR KB.

The MDSR revision phase includes two components: a KB revision mechanism and a MDSR control mechanism. The values of the initial Q-table were sent to a KB revision mechanism (Q-learning-based agent). Then, the MDSR control mechanism serves as a decision-maker for the production system. When the MDSR control mechanism executes a control decision concerning which job (or lot) should be processed, the control mechanism provides the right decision.

At the beginning of the scheduling decision period, the RL perceives the current state and sends a MDSR action to the MDSR control mechanism to initialize a MDSR KB. After all stations complete their work, production performance (i.e., reward) information is output to the Q-learning-based agent. Eventually, a KB revision mechanism updates the entire Q-table (i.e., revised MDSR KB) and decides the most suitable MDSRs for the following scheduling period. This process ends after the termination state is reached.

For example, during FAB operation, the Q-learning-based agent is assumed to perceive information on the FAB environment and is autonomously in charge of decision-making for the order release control and dispatch rule selection of wafer lots using the intrabay and stocker. The goal of the decision process is to select an action (i.e., a MDSR), where the MDSRs rely on predefined weight vectors to schedule rules. In this study, for the original Q-learning algorithm $Q(s, a)$ in the production system, $s$ is a scheduling decision point, and $a$ is a MDSR decision rule. Sections 3 and 4 will discuss each of these phases in detail.

III. INITIALIZATION PROCEDURE OF THE INITIAL MDSR KB GENERATION PHASE

A. SIMULATION-BASED TRAINING EXAMPLE GENERATION MECHANISM

A training dataset must be comprised of a comprehensive initial KB representing a wide range of possible system statuses [61]. In this research, the multi-pass simulation technique [18] is applied to generate the training dataset. The state variables of the system features are recorded at the beginning of the scheduling decision point, whereas the performance of certain MDSRs is recorded at the end of the scheduling interval. Using a simulation-based training data generation mechanism, the training dataset could provide various patterns of job arrivals and a comprehensive initial KB.

B. DATA PREPROCESSING MECHANISM

Because there is a large amount of shop floor information in the production system, using too many features can lead to overfitting of the training data, which reduces the ability of the dynamic scheduling system to generalize knowledge. Previous research [20], [60], [64] also pointed out that, in a machine learning-based dynamic scheduling system, it is crucial to select appropriate system features according to different production requirements to build a dynamic scheduling KB.

Two general methods are employed for feature selection: filter and wrapper methods [65]. The filter approach uses criteria that do not involve any machine learning algorithms, and it does not consider the effects of a selected feature subset on the performance of the machine learning algorithm. Hence, its computational efficiency is higher than that of the wrapper method. In the wrapper method, where the classifier system provides a subset of features as input, the classification error
is evaluated using an unseen dataset. As described above, a dynamic scheduling system using a MDSR mechanism cannot develop KB by using a supervised machine learning algorithm. Hence, we cannot use the wrapper method for feature selection in this research.

The Las Vegas filter (LVF) selection method [66] repeatedly generates random data subsets (denoted RD) during each iteration and then computes their evaluation measure (inconsistency rate). This method currently provides the best solution via a simultaneous probabilistic search for the most suitable set of features. The LVF algorithm [66] (Table 4) generates a RD from nf features in every iteration. If the number of features (C) of the RD is smaller than the current best (i.e., C < C_best), then the dataset denoted DS with prescribed features is assessed against the evaluation measure. If its evaluation measure is below a prespecified one (r), then C_best and RD_best are replaced by C and RD, respectively, and the new current best RD is printed. If C = C_best and the evaluation measure is satisfied, then an equally good current best is found and printed. When the LVF has looped all MAX_TRIES cycles, the process stops. The final best RD is chosen for further tests using a learning algorithm.

### C. SYSTEM STATE NUMBER DETERMINATION

In this study, the Q-learning-based agent is employed to develop a KB revision mechanism. In the Q-learning method, the agent recognizes the current state in the environment and then selects the most suitable action to take. In the classical machine learning-based dynamic scheduling approach, the number of system states is limitless. Nevertheless, the Q-table is constructed to express the knowledge perceived by the agent based on Q-learning by investigating the results of the actions taken in each state. Therefore, for Q-based learning agents, it is unreasonable to have a large number of system states. Thus, the training dataset obtained using LVF feature selection based on performance criterion o is expressed as \(oX_{LVF}\) (i.e., a tuple of \([f_{LVF}, \alpha^{next}_{LVF}]\)), where the system features obtained by the training dataset through the LVF method are expressed as \(f_{LVF}\). By applying two-level SOM analysis, \(oX_{LVF}\) can be divided into different system state classes. If the training data belong to the same class, then they can be assigned the same system state number label.

A training dataset with the same system state number label \(i (i = 1, \ldots, I)\), expressed as \(oX_{LVF}^i\), is represented as a three-tuple of \([f_{LVF}, \alpha^{next}_{LVF}]\), where \(oX_{LVF}\) expresses the subset of \(i\) (i.e., with an assigned system state label \(i\)) in \(oX_{LVF}\) obtained by clustering analysis according to performance criterion \(o\), and \(\alpha^{next}_{LVF}\) is assigned system state number label \(i\).

### D. INITIALIZATION OF THE MDSR KB

In the Q-learning-based MDSR selection mechanism, the Q-table is generated to represent the MDSR KB such that the Q-learning-based agent learns by examining the results of actions taken in each state. In this research, an action \(d^{(i)}\) is defined as the Q-table choice weight vector \(\alpha^{next}_{LVF}\) based on perceived state \(\alpha^{(i)}\) of the production system, and \(d^{(i)}\) is the class label of the \(i\)-th state of the \(j\)-th training sample (i.e., \(x^{(i)}\)) action, where \(i (i = 1, \ldots, I)\) is an index of the class label of the state, and \(j (j = 1, 2, \ldots, n)\) is an index of the training sample. Each state-action pair is associated with a Q value, namely, \(Q (\alpha^{(i)}, \alpha^{next}_{LVF})\). A training sample of a Q-table for the N_P criterion is listed in Table 5.

The initial \(Q (\alpha^{(i)}, \alpha^{next}_{LVF})\) value (i.e., the initial MDSR KB) is obtained according to the performance measures for a set of training data and the given performance criterion \(o\) (\(oX\),

### TABLE 4. LVF algorithm.

| Input: MAX_TRIES, nf, \(\alpha\) |
|----------------------------------|
| RD= randomFeatureDataSet (seed); |
| C= numberOfFeatures (RD);        |
| If (\(C > C_{best}\))           |
| RD= RD; C_{best} = C;           |
| Print_Current_Best (RD);         |
| Else if (\(C = C_{best}\) and (\(\alpha\) = \(\alpha_{LVF}\))) |
| Print_Current_Best (RD);         |
| End                               |

### TABLE 5. A Q-Table.

| State class number | Training sample number | Q value |
|--------------------|------------------------|---------|
| \(x^{(i)}\)        | \(x^{next}_{LVF}\)     | \(Q (\alpha^{(i)}, \alpha^{next}_{LVF})\) |

Y.-R. Shiue et al.: RL Approach to Dynamic Scheduling in a Product-Mix Flexibility Environment
which is calculated using equation (3):

\[
Q(o, a) = \frac{o_{\text{PM}}^o - o_{\text{PM}}^m}{o_{\text{PM}}^m - o_{\text{PM}}^m}, \quad o \in \{T_p\}
\]

\[
Q(o, a) = 1 - \frac{o_{\text{PM}}^o - o_{\text{PM}}^m}{o_{\text{PM}}^m - o_{\text{PM}}^m}, \quad o \in \{M_{\text{CT}}, N_T\}
\]

where the value of performance measure \(o_{\text{PM}}^o\) is training example \(x^o\) at the scheduling decision point based on performance criterion \(o\), \(o_{\text{PM}}^m\) is the minimum value of performance criterion \(o\) in system state class label \(i\), and \(o_{\text{PM}}^m\) is the maximum value of performance criterion \(o\) in system state class label \(i\).

IV. PROCEDURE OF THE MDSR KB REVISION PHASE: Q-LEARNING-BASED AGENT

After establishing the initial Q-table, the following issues must be resolved. Specifically, a two-level SOM can be used to decide a specific system state class number. However, it is not possible to determine how to select the most suitable MDSRs among multiple candidate MDSRs for the following scheduling period. Therefore, this study proposes a Q-learning-based agent for selecting MDSRs to solve this problem. Figure 3 shows a schematic of the online Q-learning-based agent MDSR selection process in this case study. The workflow of the Q-learning-based agent is depicted in Figure 4, and the details are provided below.

A. INITIALIZE THE SYSTEM PARAMETERS

At the end of the warm-up period, the MDSRs for the subsequent scheduling periods are executed. Some system parameters need to be determined in advance, including the governing performance criterion \(o\), the current system state \(s\), and the initial \(Q(o, a)\) value in the Q-table calculated using equation (3).

B. MDSRS SELECTION USING THE \(\epsilon\)-GREEDY POLICY

At the beginning of the scheduling period (Figure 3), MDSRs are selected by exploring and exploiting two strategies for selecting actions using the Q-learning-based agent.

Exploitation means that the agent makes the best decision based on current information, while exploration means that the agent tries something that has not been previously attempted to attain a greater reward. Exploitation promises an appropriate expected reward in each step, whereas exploration provides additional chances for identifying the global maximum reward value in the long run. One successful method for addressing this tradeoff problem is the \(\epsilon\)-greedy policy [36]. Using this mechanism, a policy (i.e., decision-making action) denoted \(\pi(o, s)\) is selected with probability \(1-\epsilon\) (where \(0 < \epsilon < 1\)), and this policy includes the best value (i.e., exploitation), as defined in equation (4); otherwise, at a low probability \(\epsilon\), an action is selected randomly.

\[
\pi(o, s) = \arg \max_{a} Q(o, s, a)\tag{4}
\]

C. UPDATE THE Q-TABLE

At the end of the scheduling interval, the learning agent attempts to maximize the Q value by interacting directly with the environment and responding to the return of rewards or penalties (i.e., the system production performance) defined using a reward function. The reward function defines the goal of maximizing the total reward; thus, the reward function basically helps the learning agent to achieve its goal.

In the current study case, according to the \(N_T\) criterion, regarding system performance, the goal is to minimize \(N_T\). After the one-step scheduling period is completed, system performance is compared with the mean performance measure in state class \(i\). If the performance measure for the one-step scheduling period output expressed as \(N_{\text{PM}_{\text{SP}}}^T\) is one standard deviation higher than the mean performance in state class \(i\) expressed as \(N_{\text{UCL}_{\text{SP}}}^T\), then the agent receives a reward of \(+1\). In addition, if \(N_{\text{PM}_{\text{SP}}}^T\) is one standard deviation lower than the mean performance in state class \(i\)
expressed as $N_T \cdot LCL_i \cdot LCL_i$, then the agent receives a reward of $-1$; otherwise, the agent receives a reward of $0$. The reward function in this research is summarized in Table 6.

### D. TERMINATION STATE

The presented Q-learning-based agent algorithm is listed in Table 7. The Q-table associated with the Q value is updated, and the most suitable MDSRs are determined for the subsequent scheduling period through the end of the termination state (i.e., the end of the simulation run).

### V. EXPERIMENT AND RESULTS

#### A. SIMULATION MODEL CONSTRUCTION AND TRAINING EXAMPLE GENERATION

To verify the proposed method, a discrete-event simulation model is used to generate a training dataset. The simulation model is developed and executed by Tecnomatix Plant Simulation (2009) [67] running on an Intel i7-4790 3.60 GHz processor with 8 GB of random access memory and Windows 10 operating system.

Several parameters are determined in a preliminary simulation run. In the FMS study case, the time between job arrivals is exponentially distributed with a mean of 31 minutes. The due date of each job is randomly assigned from 6 to 10 times the total processing time and uniformly distributed. Table 8 lists the mix ratios of the five products that experience various conditions of unbalanced machine loads and bottleneck shifting. The part-type mix ratios vary every 20,000 minutes.

In the FAB case study, the time between job arrivals is exponentially distributed with a mean of 95 minutes, which is the same as that used in an earlier study [58]. The due date of each job is randomly assigned from 6 to 10 times the total processing time and uniformly distributed. Table 9 lists the six lot-type mix ratios used to generate the training examples. The part-type mix ratios vary every 50,000 minutes.

To generate a large number of training datasets in the FMS, we use 80 random seeds to generate 80 different job arrival patterns. The warm-up time for each run is 5,000 minutes, followed by 100 multi-pass scheduling periods. The scheduling period of one multi-pass simulation is 1,000 minutes.

In the FAB case, we use 80 random seeds to generate 80 different job arrival patterns. The warm-up time for each run is 144,000 minutes (10 days), followed by 100 multi-pass scheduling periods. The scheduling period of one multi-pass simulation is 5,000 minutes. Hence, a total of 8,000 training datasets were collected for the two case studies. Here, the warm-up time is determined in a preliminary simulation run based on the model reaching a steady state and a machine utilization of approximately 80% [68].

#### B. FEATURE SELECTION AND SYSTEM STATE DETERMINATION IN MDSR KB INITIALIZATION

Figure 2 shows that the system attributes must be selected before building an initial MDSR KB. The employed LVF feature selection algorithm is coded in MATLAB 2016 [69]. Here, the inconsistency rate is set to 0.50. In this study, for each production performance criterion, a two-level SOM algorithm is applied to cluster the training instances to set up a KB class label. The two-level SOM algorithm is coded using the MATLAB Neural Network Toolbox [70]. Table 10 lists the search using $k$ values from 2 to 10 for each performance criterion DB index [71]. In FMS, the minimum DB indexes selected according to the $T_P$, $M_CT$, and $N_T$ criteria were 9, 10, and 6, respectively. In FAB, the minimum DB indexes selected according to the $T_P$, $M_CT$, and $N_T$ criteria were 9, 8, and 8, respectively. According to the results in Table 10, to satisfy the $T_P$, $M_CT$, and $N_T$ performance criteria, the FMS...
TABLE 10. DB index of each performance criterion.

| Case | Criterion | No. of cluster |
|------|-----------|----------------|
| 1    | 2         | 3              |
| 4    | 5         | 6              |
| 7    | 8         | 9              |
| 7    | 10        |                |

Y.-R. Shiue et al.: RL Approach to Dynamic Scheduling in a Product-Mix Flexibility Environment

TABLE 11. Part-type probability distribution in product-mix environments for experiment verification in FMS.

| Part ID | Part-type probability distribution |
|---------|------------------------------------|
| Type 1  | Type 2  | Type 3  | Type 4  | Type 5  |
| 1       | 1.20    | 0.80    | 0.60    | 0.40    | 0.20    |
| 2       | 0.80    | 0.60    | 0.40    | 0.20    | 0.10    |
| 3       | 0.40    | 0.20    | 0.10    |          |          |

C. SIMULATION EXPERIMENT VERIFICATION

A Q-learning-based agent encoded using the Tecnomatix Plant SimTalk simulation language (2009) [67] is linked to a C program (by using the two-level SOM algorithm coded in MATLAB for determining the system state class number) to investigate the utility of the presented RL-based dynamic scheduling approach by the MDSR mechanism under different production system circumstances.

We verified that the proposed RL-based dynamic scheduling system using the MDSR mechanism is effective based on various product-mix flexibility environments. Three different product-mix ratios are designed in FMS and FAB cases (listed in Tables 11 and 12, respectively) for online operation. A stream of arriving jobs is generated for a simulation period of 300,000 minutes and one year for the FMS and the FAB cases, respectively. In this study, we want to prove that the proposed RL-based dynamic scheduling method using the MDSR mechanism is superior to the fixed-decision scheduling method, the machine learning classification approach (i.e., SDSR), and the classic MDSR selection mechanism (which does not include a KB revision mechanism).

TABLE 12. Part-type probability distribution in product-mix environments for experiment verification in FAB.

| Part ID | Part-type probability distribution |
|---------|------------------------------------|
| Type 1  | Type 2  | Type 3  | Type 4  | Type 5  |
| 1       | 1.20    | 0.80    | 0.60    | 0.40    | 0.20    |
| 2       | 0.80    | 0.60    | 0.40    | 0.20    | 0.10    |
| 3       | 0.40    | 0.20    | 0.10    |          |          |

TABLE 13. Simulation results in FMS.

| Scheduling strategy | Mean | Standard deviation | Mean | Standard deviation | Mean | Standard deviation |
|---------------------|------|--------------------|------|--------------------|------|--------------------|
| RL-MDSR             | 13016.9 | 76.3               | 81.0 | 2988               | 1261 |                    |
| Classical MDSR      | 1284.6 | 185.6              | 716  | 148               | 4120 | 1893               |
| GA-SVM              | 12957.8 | 168.1              | 799  | 174               | 4041 | 1038               |
| DS                  | 12615.9 | 179.8              | 1410 | 336               | 10070 | 1440               |
| EDSS                | 12670.6 | 156.6              | 954  | 209               | 9625 | 1772               |
| SPT                 | 12712.9 | 167.0              | 1095 | 251               | 8408 | 1554               |
| SQO                 | 12738.1 | 166.5              | 1905 | 415               | 7864 | 1594               |
| SRPT                | 12556.4 | 170.1              | 1757 | 567               | 11693 | 534               |

Here, because FAB is a large complex manufacturing system, it is unlikely to use a single control decision rule (i.e., dispatch rule) when making scheduling decisions; thus, SDSR cannot be employed in a FAB environment.

In this study, we select a suitable MDSR decision rule by using the ε-greedy policy. The Q-learning-based agent uses the same common parameter settings (α = 0.1, γ = 0.9, and ε = 0.1) for all of the experimental runs. In the FMS case study, the fixed-decision scheduling approach includes five heuristic dispatch rules: DS, EDD, SIO, SPT, and SRPT. In the FAB case study, the fixed-decision scheduling approach includes five methods: WL+DS, WL+EDD, WL+SIO, WL+SPT, and WL+SRPT. Here, for WL+DS, at the beginning of each scheduling period, WL decision rules are used for wafer order release control, and DS scheduling rules are used to select intrabay and stocker wafer lots. In the FMS case study, the SDSR approach constructs a dynamic scheduling KB using the GA+SVM approach because of its superiority [20]. Here, GA (genetic algorithm) [72] is a wrapper method for system feature selection. For both FMS and FAB case studies, the classical MDSR approach constructs a dynamic scheduling KB using a SOM neural network [73], which is based on earlier research [74], [75].

Tables 13 and 14 display the mean and standard deviations of 30 simulation runs with 30 random seeds in the FMS and FAB cases, respectively. Based on the measurement results of all performance criteria, the proposed approach achieves excellent results due to its high efficiency.

Since the same system scenarios were implemented using common random seeds, the paired sample t-test is used to determine whether the proposed RL-based dynamic scheduling method based on the MDSR selection mechanism is superior to the fixed-decision scheduling and classical...
According to the research results, the proposed RL-based dynamic scheduling method using the MDSR mechanism is considerably superior to its three peers. Hence, the proposed RL-based dynamic scheduling method using the MDSR mechanism is considerably superior to its three peers.

VI. CONCLUSION AND FUTURE WORKS

In this research, we present a RL-based dynamic scheduling method with the MDSR mechanism for establishing a dynamic scheduling system in a complex product-mix ratio environment. The following inferences can be made according to the research results:

- A product-mix ratio flexibility environment experiences bottlenecks and capacity-constraining resource issues. In this situation, the scheduling KB should be dynamic, and therefore, a knowledge revision mechanism should be established to monitor the crucial changes occurring in the production system. This mechanism can solve the problem of bottleneck shifting and unbalanced machine loads caused by resource capacities.

- This research proposes a RL-based dynamic scheduling method that applies the MDSR selection mechanism to effectively respond to product-mix ratio variations in FMS and FAB systems. The method is applicable to the operation of smart factory dynamic scheduling.

- The proposed RL-based dynamic scheduling method uses a dynamic and multi-pass approach for deciding MDSRs, which is applied in accordance with the status of the production system at the beginning of the scheduling interval, and the system then decides the most suitable MDSRs for the next scheduling interval.

- Although generating training datasets and learning processes to obtain RL-based dynamic scheduling by applying a MDSR mechanism KB is often time consuming compared to a fixed-scheduling approach, the proposed approach can perform better than the fixed-scheduling approach, the machine learning classification approach, and the classical MDSR selection mechanism in the long run.

The following potential issues for future research have been recognized based on the results of this research:

- In this study, a MDSR KB is initialized through a system state number determination step (by the two-level SOM algorithm), which could reduce the time required by traditional machine learning methods to generate a training dataset. Although this approach is feasible, semi-supervised learning [76], which uses unlabeled data for training, can typically be used with a small amount of labeled data and a large amount of unlabeled data. This idea can be examined in future studies.

- Recently, some studies [77], [78] have used deep learning for dynamic scheduling problems. The long short-term memory (LSTM) [79] is a recurrent neural network (RNN) architecture used in deep learning [80]. In the RNN, the output from the last step is fed as input in the current step, which can address the problem of the long-term dependencies of the RNN. As the gap length increases, the RNN performance efficiency declines. The LSTM can, by default, retain information for a long period of time, and the LSTM can process not only single data points (such as images) but also entire data sequences (as in speech recognition, mathematical finance, weather forecasting, and shop floor control). These features can spur future studies of MDSRs in intelligent manufacturing research issues using the LSTM. Hence, we intend to study this concept in the future.

REFERENCES

[1] J. Bengtsson and J. Olhager, “The impact of the product mix on the value of flexibility,” Omega, vol. 30, no. 4, pp. 265–273, Aug. 2002, doi: 10.1016/S0305-0483(02)00034-8.

[2] M. V. Martin and K. Ishii, “Design for variety: Developing standardized and modularized product platform architectures,” Res. Eng. Des., vol. 13, no. 4, pp. 213–235, Nov. 2002, doi: 10.1007/S00163-002-0020-2.

[3] T. Jannrus, C.-F. Chien, M. Gen, and K. Sethanan, “Hybrid particle swarm optimization combined with genetic operators for flexible job-shop scheduling under uncertain processing time for semiconductor manufacturing,” IEEE Trans. Semicond. Manuf., vol. 31, no. 1, pp. 32–41, Feb. 2018, doi: 10.1109/TSM.2017.2758380.

[4] T. Meng, Q.-K. Pan, and H.-Y. Sang, “A hybrid artificial bee colony algorithm for a flexible job shop scheduling problem with overlapping in operations,” Int. J. Prod. Res., vol. 56, no. 16, pp. 5278–5292, Aug. 2018, doi: 10.1080/00207543.2018.1467575.

[5] K. Gao, Z. Cao, L. Zhang, Z. Chen, Y. Han, and Q. Pan, “A review on swarm intelligence and evolutionary algorithms for solving flexible job-shop scheduling problems,” IEEE/CAA J. Automatica Sinica, vol. 6, no. 4, pp. 904–916, Jul. 2019, doi: 10.1109/JAS.2019.1911540.

[6] A. Shahzad and N. Mebarki, “Data mining based job dispatching using hybrid simulation-optimization approach for shop scheduling problem,” Eng. Appl. Artif. Intell., vol. 25, no. 6, pp. 1173–1181, Sep. 2012, doi: 10.1016/j.engappai.2012.04.001.

---

**TABLE 14. Simulation results in FAB.**

| Scheduling strategy | $T_{e}$ (Mean) | $T_{e}$ (Standard deviation) | $N_{e}$ (Mean) | $N_{e}$ (Standard deviation) |
|---------------------|----------------|-----------------------------|----------------|-----------------------------|
| RL-MDSR             | 5995.1         | 76.9                        | 11950          | 3477                        |
| Classical MDSR      | 5259.5         | 552.2                       | 24112          | 4404                        |
| WI-DS               | 5187.1         | 136.3                       | 21983          | 3546                        |
| WI-EDD              | 5242.2         | 127.5                       | 21289          | 3912                        |
| WI-SPT              | 5342.1         | 136.6                       | 26832          | 3463                        |
| WI-SO               | 5097.3         | 146.1                       | 24849          | 2628                        |
| WI-SRFT             | 5377.9         | 132.8                       | 20165          | 3398                        |

**TABLE 15. The paired sample t test under three production criteria in both cases.**

| Study case | Performance criteria | $T_{e}$ | $M_{T_{e}}$ | $N_{e}$ | $M_{N_{e}}$ | $P$-value |
|------------|----------------------|---------|------------|--------|------------|-----------|
| Classical  | MDSR                 |         |            |        |            |           |
| GA-SVM     | (WI-DS)              | 0.006   | 0.003      | 0.000  | 0.000      | 0.000     |
|            | (WI-EDD)             | 0.050   | 0.025      | 0.000  | 0.000      | 0.000     |
|            | (WI-SPT)             | 0.000   | 0.007      | 0.000  | 0.000      | 0.000     |
|            | (WI-SO)              | 0.000   | 0.000      | 0.000  | 0.000      | 0.000     |
|            | (WI-SRFT)            | 0.027   | 0.000      | 0.000  | 0.000      | 0.000     |

MDSR methods. The null hypothesis means that the mean values obtained by all of the scheduling approaches are equal. Tables 13 and 14 display simulation experimental results. The values of the paired sample t-tests under three performance criteria in FMS and FAB are summarized in Table 15; the null hypothesis is rejected at a significance level of 95% for all scheduling approaches. Hence, the proposed RL-based dynamic scheduling method using the MDSR mechanism is considerably superior to its three peers.
[7] R. G. Qiu and M. Zhou, “Mighty MESS: state-of-the-art and future manufacturing execution systems,” IEEE Robot. Autom. Mag., vol. 11, no. 1, pp. 19–25, Mar. 2004, doi: 10.1109/MRA.2004.1275947.

[8] B. S. de Ugarte, A. Artiba, and R. Pellerin, “Manufacturing execution system—A literature review,” Prod. Planning Control, vol. 20, no. 6, pp. 525–539, Jun. 2009, doi: 10.1080/09537280902936313.

[9] R. Y. Zhong, Y. Y. Dai, T. Qu, G. J. Hu, and G. Q. Huang, “RFID-enabled real-time manufacturing execution system for mass-customization production,” Robot. Comput. Integr. Manuf., vol. 29, no. 2, pp. 283–292, Apr. 2013, doi: 10.1016/j.rcim.2012.08.001.

[10] Y. Zhang, W. Wang, N. Wu, and C. Qian, “IoT-f-enabled real-time production performance analysis and exception diagnosis model,” IEEE Trans. Autom. Sci. Eng., vol. 13, no. 3, pp. 1318–1332, Jul. 2016, doi: 10.1109/TASE.2015.2497800.

[11] J. Leng and P. Jiang, “Dynamic scheduling in RFID-driven discrete manufacturing system by using multi-layer network metrics as heuristic information,” J. Intell. Manuf., vol. 30, no. 3, pp. 979–994, Mar. 2019, doi: 10.1007/s10845-017-1301-y.

[12] D. A. Rossiti, F. Tohmé, and M. Frutos, “Industry 4.0: Smart scheduling,” Int. J. Prod. Res., vol. 57, no. 12, pp. 3802–3813, Jun. 2019, doi: 10.1080/00207543.2018.1504248.

[13] C. Qian, Y. Zhang, C. Jiang, and S. Pan, “A real-time data-driven collaborative mechanism in fixed-position assembly systems for smart manufacturing,” Robot. Comput. Integr. Manuf., vol. 61, Feb. 2020, Art. no. 101841, doi: 10.1016/j.rcim.2019.101841.

[14] Y. Z. Son, H. Rodríguez-Rivera, M. A. Rico, and R. A. Wysk, “A multi-pass simulation-based, real-time scheduling and shop floor control system,” Trans. Soc. Comput. Simul., vol. 16, no. 4, pp. 159–172, Dec. 1999.

[15] Y.-R. Shiue, K.-C. Lee, and C.-T. Su, “Real-time scheduling for a smart manufacturing system based on a composite rule set,” Appl. Sci., vol. 7, no. 10, pp. 2281–2294, Dec. 2012, doi: 10.3390/app7102281.

[16] J. Leng and P. Jiang, “Dynamic scheduling in RFID-driven discrete manufacturing system by using multi-layer network metrics as heuristic information,” J. Intell. Manuf., vol. 30, no. 3, pp. 979–994, Mar. 2019, doi: 10.1007/s10845-017-1301-y.

[17] D. A. Rossiti, F. Tohmé, and M. Frutos, “Industry 4.0: Smart scheduling,” Int. J. Prod. Res., vol. 57, no. 12, pp. 3802–3813, Jun. 2019, doi: 10.1080/00207543.2018.1504248.

[18] C. Qian, Y. Zhang, C. Jiang, and S. Pan, “A real-time data-driven collaborative mechanism in fixed-position assembly systems for smart manufacturing,” Robot. Comput. Integr. Manuf., vol. 61, Feb. 2020, Art. no. 101841, doi: 10.1016/j.rcim.2019.101841.

[19] B. Jang, M. Kim, G. Harperimana, and J. W. Kim, “Q-learning algorithms: A comprehensive classification and applications,” IEEE Access, vol. 7, pp. 133653–133667, 2019, doi: 10.1109/ACCESS.2019.2941229.

[20] F. Zhang, J. Leitner, M. Milford, B. Upcroft, and P. Corke, “Towards vision-based deep reinforcement learning for robotic motion control,” 2015, arXiv:1511.03791. [Online]. Available: https://arxiv.org/abs/1511.03791

[21] J. Jiang, H. Huang, and Z. Ding, “Path planning for intelligent robots based on deep Q-learning with experience replay and heuristic knowledge,” IEEE/CAA J. Automatica Sinica, early access, Sep. 2019, doi: 10.1109/JAS.2019.1911732.

[22] J. Si and Y.-T. Wang, “Online learning control by association and reinforcement,” IEEE Trans. Neural Netw. Learn. Syst., vol. 12, no. 2, pp. 264–276, Mar. 2001, doi: 10.1109/72.914523.

[23] B. Kiumarsi, K. G. Vamvoudakis, H. Modares, and F. L. Lewis, “Optimal and autonomous control using reinforcement learning: A survey,” IEEE Trans. Neural Netw. Learn. Syst., vol. 29, no. 6, pp. 2042–2062, Jun. 2018, doi: 10.1109/TNNLS.2017.2773458.

[24] R. C. Plotnick, J. K. Schrader, N. Stricker, and G. Lanza, “Design, implementation and evaluation of reinforcement learning for an adaptive order dispatching in job shop manufacturing systems,” Procedia Comput. Sci., vol. 50, pp. 234–239, 2019, doi: 10.1016/j.procs.2019.03.041.

[25] A. Kühler, L. Schätzer, N. Stricker, and G. Lanza, “Autonomous order dispatching in the semiconductor industry using reinforcement learning,” Procedia CIRP, vol. 81, pp. 234–239, 2019, doi: 10.1016/j.procs.2019.03.041.

[26] H.-N. Dai, H. Wang, G. Xu, J. Wan, and M. Imran, “Big data analytics for manufacturing Internet of Things: Opportunities, challenges and enabling technologies,” Enterprise Inf. Syst., pp. 1–25, Jun. 2019, doi: 10.1080/1751758X.2019.1633689.

[27] G. C. J. H. Watkins and P. Dayan, “Q-learning,” Mach. Learn., vol. 8, nos. 3–4, pp. 279–292, 1992, doi: 10.1023/A:1022176721222.

[28] Y.-C. Wang and J. M. Usher, “Application of reinforcement learning for agent-based production scheduling,” Eng. Appl. Artif. Intell., vol. 18, no. 1, pp. 73–82, Feb. 2005, doi: 10.1016/j.engappai.2004.08.018.

[29] H. Cho and R. A. Wysk, “A robust adaptive scheduler for an intelligent workstation controller,” Int. J. Prod. Res., vol. 31, no. 4, pp. 771–789, Apr. 1993, doi: 10.1080/00207549308957656.
C.-C. Lee and J. T. Lin, “Deadline prediction and avoidance based on Petri nets for controller digitalized automated guided vehicle systems,” *Int. J. Prod. Res.*, vol. 33, no. 12, pp. 3249–3265, Dec. 1995, doi: 10.1080/00207540508904872.

N. Wu and W. Zeng, “Deadline avoidance in an automated guided vehicle system using a colored Petri net model,” *Int. J. Prod. Res.*, vol. 40, no. 1, pp. 223–238, Jan. 2002, doi: 10.1080/00207540110073037.

K. Xing, L. Han, M. Zhou, and F. Wang, “Deadline-free genetic scheduling algorithm for automated manufacturing systems based on deadlock control policy,” *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 42, no. 3, pp. 603–615, Jun. 2012, doi: 10.1109/TSMCB.2011.2170678.

M. Montazeri and L. N. Van Wassenhove, “Analysis of scheduling rules for an FMS,” *Int. J. Prod. Res.*, vol. 28, no. 4, pp. 785–802, Apr. 1990, doi: 10.1080/00207549008942754.

E. Campbell and J. Ammenheuser, 300 mm Factory Layout and Material Handling Modeling: Phase II Report, Technology Transfer Document International SEMATECH, 2000.

L. M. Wein, “Scheduling semiconductor wafer fabrication,” *IEEE Trans. Semicond. Manuf.*, vol. SM-1, no. 3, pp. 115–130, Aug. 1988, doi: 10.1109/66.4384.

S. C. H. Lu, D. Ramaswamy, and P. R. Kumar, “Efficient scheduling policies to reduce mean and variance of cycle-time in semiconductor manufacturing plants,” *IEEE Trans. Semicond. Manuf.*, vol. 7, no. 3, pp. 374–388, Aug. 1994, doi: 10.1109/66.311341.

B.-W. Hsieh, C.-H. Chen, and S.-C. Chang, “Scheduling semiconductor wafer fabrication by using ordinal optimization-based simulation,” *IEEE Trans. Robot. Autom.*, vol. 17, no. 5, pp. 599–608, Oct. 2001, doi: 10.1109/70.964661.

S. C. Park, N. Ramam, and M. J. Shaw, “Adaptive scheduling in dynamic flexible manufacturing systems: A dynamic rule selection approach,” *IEEE Trans. Robot. Autom.*, vol. 13, no. 4, pp. 486–502, Aug. 1997, doi: 10.1109/70.611301.

C.-C. Chen, Y. Yih, and Y.-C. Wu, “Auto-bias selection for developing learning-based scheduling systems,” *Int. J. Prod. Res.*, vol. 37, no. 9, pp. 1987–2002, Jun. 1999, doi: 10.1080/0020754991908789.

Y. Arzi and L. Iaroslavitz, “Operating an FMC by a decision-tree-based adaptive production control system,” *Int. J. Prod. Res.*, vol. 38, no. 3, pp. 675–697, Feb. 2000, doi: 10.1080/0020754001893656.

Y.-R. Shiue and R.-S. Guh, “Learning-based multi-pass adaptive scheduling for a dynamic manufacturing cell environment,” *Robot. Comput.-Integr. Manuf.*, vol. 22, no. 3, pp. 203–216, Jul. 2006, doi: 10.1016/j.rcim.2005.03.004.

J. Vesanto and E. Alhoniemi, “Clustering of the self-organizing map,” *IEEE Trans. Neural Netw.*, vol. 11, no. 3, pp. 586–600, May 2000, doi: 10.1109/72.846731.

C.-T. Su and Y.-R. Shiue, “Intelligent scheduling controller for shop floor control systems: A hybrid genetic algorithm/decision tree learning approach,” *Int. J. Prod. Res.*, vol. 41, no. 12, pp. 2619–2641, Jan. 2003, doi: 10.1080/0020754031000090612.

H. Liu and H. Motoda, Feature Selection for Knowledge Discovery and Data Mining, New York, NY, USA: Springer, 2012.

H. Liu and R. Setiono, “A probabilistic approach to feature selection—A filter solution,” in *Proc. ICML*, 1996, pp. 319–327.

Technomatix Plant Simulation, “Technomatix plant simulation 9.0 user guide,” Technomatix Technol., Kingdom Prussia, Berlin, Germany, 2009.

P. S. Mahajan and R. G. Ingalls, “Evaluation of methods used to detect warm-up period in steady state simulation,” in *Proc. Winter Simulation Conf.*, vol. 1, 2004, pp. 663–671, doi: 10.1109/WSC.2004.1371374.

MATLAB Release 2016b, The MathWorks, Natick, MA, USA, 2016.

H. Demuth, M. Beale, and M. Hagan, “Neural network toolbox 5 user’s guide,” MathWorks, New Delhi, India, Tech. Rep., 2006.

D. L. Davies and D. W. Bouldin, “A cluster separation measure,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-1, no. 2, pp. 224–227, Feb. 1979, doi: 10.1109/TPAMI.1979.476609.

D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Reading, MA, USA: Addison-Wesley, 1989.

T. Kohonen, *Self-Organizing Maps*, Berlin, Germany: Springer-Verlag, 2001, p. 528.

R.-S. Guh, Y.-R. Shiue, and T.-Y. Tseng, “The study of real-time scheduling by an intelligent multi-controller approach,” *Int. J. Prod. Res.*, vol. 49, no. 10, pp. 2977–2997, May 2011, doi: 10.1080/00207541003794884.

Y.-R. Shiue, R.-S. Guh, and K.-C. Lee, “Study of SOM-based intelligent multi-controller for real-time scheduling,” *Appl. Soft Comput.*, vol. 11, no. 8, pp. 4569–4580, Dec. 2011, doi: 10.1016/j.asoc.2011.07.022.

O. Chapelle, B. Schlkopf, and A. Zien, *Semi-Supervised Learning*, Cambridge, MA, USA: MIT Press, 2010, p. 528.

H. Zhou, B. E. Lim, and S. Lee, “Deep learning-based dynamic scheduling for semiconductor manufacturing with high uncertainty of automated material handling system capability,” *IEEE Trans. Semicond. Manuf.*, vol. 33, no. 1, pp. 13–22, Feb. 2020, doi: 10.1109/TSM.2020.2965293.

H. Zhu, M. Li, Y. Tang, and Y. Sun, “A deep-reinforcement-learning-based optimization approach for real-time scheduling in cloud manufacturing,” *IEEE Access*, vol. 8, pp. 9987–9997, 2020, doi: 10.1109/ACCESS.2020.2964955.

S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, Cambridge, MA, USA: MIT Press, 2016.