Computers & Industrial Engineering 125 (2018) 604–614

Contents lists available at ScienceDirect

Computers & Industrial Engineering

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

Real-time scheduling for a smart factory using a reinforcement learning approach

Yeou-Ren Shiue[a], Ken-Chuan Lee[b], Chao-Ton Su[b,*]

[a] Department of Management Engineering, Fujian Business University, Fuzhou 350012, PR China
[b] Department of Industrial Engineering and Engineering Management, National Tsing Hua University, Hsinchu, Taiwan, ROC

ARTICLE INFO

Keywords:
Machine learning
Q-learning
Real-time scheduling
Reinforcement learning
Shop floor control

ABSTRACT

Previous studies of the real-time scheduling (RTS) problem domain indicate that using a multiple dispatching rules (MDRs) strategy for the various zones in the system can enhance the production performance to a greater extent than using a single dispatching rule (SDR) over a given scheduling interval for all the machines in the shop floor control system. This approach is feasible but the drawback of the previously proposed MDRs method is its inability to respond to changes in the shop floor environment. The RTS knowledge base (KB) is not static, so it would be useful to establish a procedure that maintains the KB incrementally if important changes occur in the manufacturing system. To address this issue, we propose reinforcement learning (RL)-based RTS using the MDRs mechanism by incorporating two main mechanisms: (1) an off-line learning module and (2) a Q-learning-based RL module. According to various performance criteria over a long period, the proposed approach performs better than the previously proposed MDRs method, the machine learning-based RTS using the SDR approach, and heuristic individual dispatching rules.

1. Introduction

Industry 4.0, also called "Smart Factory," aims to increase factory productivity and the efficient utilization of resources in real time (Herrmann, Pentek, & Otto, 2016; Wang, Wan, Li, & Zhang, 2016). These objectives are achieved via flexible event-driven reactions to changes in the factory environment, resource allocation, scheduling, optimization, and control in real time. Most of the "Smart Factory" concepts share the attributes of cyber-physical systems (CPS) for monitoring physical processes by creating a virtual copy of the physical world and making decentralized decisions (Lee, Bagheri, & Kao, 2015).

CPS is defined as a transformative technology for managing interconnected systems according to their physical assets and computational capabilities, and recent developments have improved the availability and affordability of sensors, data acquisition systems, and computer networks (Lee, 2008; Wolf, 2009). The competitive nature of current industry is forcing more factories to implement high-tech methods. Thus, the increasing use of sensors, RFID, and networked machines has resulted in the continuous generation of high volume data known as Big Data (Lee, Kao, & Yang, 2014; Lee, Lapira, Bagheri, & Kao, 2013). In this environment, CPS can be developed further to manage Big Data and exploit the interconnectivity among machines to fulfill the goal of producing intelligent, resilient, and self-adaptable machines. Furthermore, by integrating CPS with production, logistics, and services in current industrial practices, it will be possible to transform current factories into Industry 4.0 factories with significant economic potential. This is why it is timely and crucial to consider adaptive scheduling and control (i.e., real-time scheduling; RTS) for dynamic manufacturing environments as key research issues in CPS production management (Goryachev et al., 2013; Kück, Ehm, Freitag, Frazzon, & Pimentel, 2016).

RTS employs different scheduling rules in a dynamic and multi-pass manner in order to select the best scheduling strategy among the feasible alternatives at each decision point over a series of scheduling periods, thereby meeting the shop floor performance criteria (Son, Rodriguez-Rivera, & Wysk, 1999). According to previous studies, RTS involves two main approaches (Priore, Gómez, Pino, & Rosillo, 2014): the multi-pass simulation approach (Ishii & Talavage, 1994; Wu & Wysk, 1989) and machine learning approach (Metan, Sabuncuoglu, & Pierreval, 2010; Olafsson & Li, 2010; Shiue, 2009; Shiue, Guh, & Tseng, 2012). Multi-pass simulations are used to evaluate candidate scheduling rules and select the best strategy based on simulated information, such as the current system status and the management goals for each scheduling period. However, the multi-pass simulation approach is

* Corresponding author at: Department of Industrial Engineering and Engineering Management, National Tsing Hua University, No. 101, Section 2, Kuang-Fu Road, Hsinchu 30013, Taiwan, ROC.
E-mail addresses: yrshiue@gmail.com (Y.-R. Shiue), kinwind@gapp.nthu.edu.tw (K.-C. Lee), ctsu@mx.nthu.edu.tw (C.-T. Su).

https://doi.org/10.1016/j.cie.2018.03.039

Available online 26 March 2018

inappropriate for shop floor control because it requires intensive computational effort to select the best scheduling rule for each scheduling period. In the machine learning approach for RTS, a set of training examples generated by system simulations are used to determine the best scheduling rule for each possible system state. However, the machine learning approach employed for collecting training examples and learning processes in order to acquire an RTS knowledge base (KB) tends to be time consuming and relatively slow. A KB has the advantage of yielding fast and acceptable solutions to allow the system to make decisions in real time, and it can conform to the operational characteristics of a dynamic manufacturing environment (Priore et al., 2014). Previous studies (Shiue, Guh, & Lee, 2012) defined three major machine learning approaches for constructing an RTS system KB: artificial neural networks (ANNs) (Rumelhart, Hinton, & Williams, 1986), decision tree (DT) learning (Quinlan, 1993), and support vector machines (SVMs) (Vapnik, 2000).

According to previous studies, two strategies can be used to determine the scheduling rules in an RTS system: a single dispatching rule (SDR) and multiple dispatching rules (MDRs) for a manufacturing cell. The SDR usually assigns an individual heuristic scheduling rule to all machines in a system during a given scheduling interval (i.e., scheduling period), whereas the MDRs assign different scheduling rules (i.e., scheduling decision variables) to all machines in a system. In the following, we refer to this method as an intelligent multi-controller. Fig. 1 illustrates the role of the RTS MDRs mechanism in a flexible manufacturing system (FMS) case study. For the F1, F2, F3, and load/unload stations, the MDRs method selects the SPT, SRPT, DS, and EDD dispatching rules, respectively, as the scheduling decision variables for job selection in the next scheduling period. Ishii and Talavage (1994) proposed a search algorithm that employs MDRs in bottleneck machines by using predictions based on a multi-pass simulation. Their results showed that the MDRs strategy can improve the performance of an FMS by up to 15.9% compared with the best result obtained using the SDR strategy. However, their approach is not highly suitable an RTS system that uses the machine learning approach.

The classical machine learning approach builds a RTS KB via the MDRs mechanism, and its main disadvantage is that the classes (dispatching decision rules) to which the training examples are assigned must be predefined. For example, for a given set of system attributes, the best dispatching decision rule for each decision variable can be determined after a simulation is run for each dispatching rule. The resulting MDRs are considered as a class. However, this process becomes intolerably time consuming because the rules must be determined for each period (Kim, Min, & Yih, 1998). Furthermore, the local approach, such as using DT learning or SVMs, does not satisfy the global objective function (i.e., the overall production performance of the shop floor). Thus, although the best decision rule can be determined for each scheduling decision variable, the combination of all the decision rules may not simultaneously satisfy the global objective function.

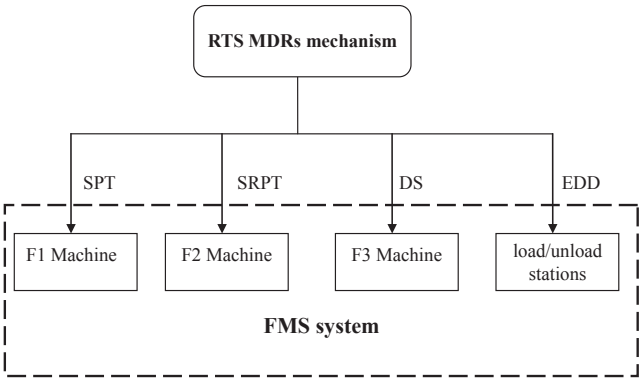

Fig. 1. The role of RTS MDRs mechanism in this study.

Guh, Shiue, and Tseng (2011) constructed an RTS KB using an MDRs selection mechanism based on a self-organizing map (SOM) neural network (Kohonen, 2001), which can overcome the long training time problem that affects the classical machine learning approach in the training example generation phase. They showed that over a long period, this approach provides better system performance than machine learning-based RTS using the SDR approach and heuristic individual dispatching rules according to various performance criteria. This approach is feasible but the main drawback of this method is its inability to respond to changes in the shop floor environment. However, the RTS KB is not static, so it would be useful to establish a procedure that maintains the KB incrementally if important changes occur in the manufacturing system.

A possible solution may incorporate a reinforcement learning (RL) mechanism that can learn to select appropriate actions for achieving its goals via interactions with the system environment and by responding to receipts for rewards or penalties based on the impact of each action (Stutton & Barto, 1998; Shahrabi, Adibi, & Mahootchi, 2017). Q-learning (Watkins & Dayan, 1992) is a form of model-free RL that provides agents with the capacity to learn to act optimally in Markovian domains by experiencing the consequences of actions, but without requiring them to build maps of the domains. Wang and Usher (2005) found that Q-learning works well for a single machine dispatching rule selection problem when used by a learning agent to select various dispatching rules according to different system criteria, and the results of their study may inspire future applications of RL techniques to the RTS problem.

Based on the studies mentioned above, in order to develop the MDRs mechanism in RTS, the RTS should be capable of updating and maintaining the KB via RL during operations to allow responses to change in the system operating conditions. Hence, using a Q-learning RL agent to refine the RTS KB is an important research issue. In this study, we develop an RL-based RTS using the MDRs mechanism. Implementation results of a study case experiment showed that the production performance has been greatly improved compared with classical SDR.

The remainder of this paper is divided into six sections. In Section 2, we present the theoretical background related to our proposed off-line learning module for determining the system state number for a Q-learning RL agent using the Self-Organizing Map (SOM) algorithm (Kohonen, 2001), and we also introduce an RL module that uses a Q-learning algorithm. In Section 3, we formulate the RTS problem using the MDRs mechanism and state the research objectives. In Section 4, we describe the method for the proposed RL-based RTS using the MDRs mechanism. In Section 5, we present the results of a study case experiment as well as analyzing the proposed RL approach and other approaches. Finally, in Section 6, we give our conclusions as well as providing a summary of this study and some suggestions for future research.

2. Theoretical background

2.1. SOM neural networks

SOM networks (Kohonen, 2001) are used widely for data mining because they are a convenient visual tool. Unlike other ANN approaches, the SOM network performs unsupervised learning, i.e., the processing units in the network adjust their weights through lateral feedback connections. The more common approach to ANNs requires supervised learning, i.e., a set of training samples is provided as the input and the output is compared with a known result. Deviations from the correct result lead to adjustments of the weights attached to the processing units. The networks are considered to have been trained when they work satisfactorily for the test cases. By contrast, an SOM network does not require prior knowledge of the expected result. The nodes in the network converge to form clusters or groups of entities

with similar properties. The number and composition of the clusters can be represented graphically based on the output distribution generated by the learning process.

Three basic steps involved in the application of the SOM algorithm after initialization comprise sampling, similarity matching, and updating. These three steps are repeated until the feature map is completed. The SOM algorithm is summarized as follows.

1. *Initialization.* Choose random values for the initial weight vectors $\mathbf{w}_j(0)$, which must be different for $j = 1, 2, \ldots, J$, where J is the number of neurons in the lattice. Low weight values are generally preferable.

Another way of initializing the algorithm is to select the weight vectors $\{\mathbf{w}_j(0)\}_{j=1}^{J}$ randomly from the available set of input vectors $\{\mathbf{x}_i\}_{i=1}^{N}$.

2. *Sampling.* Select a training example $\mathbf{x}$ from the input space with a certain probability, where the vector $\mathbf{x}$ represents the activation pattern applied to the lattice.
3. *Similarity Matching.* Find the best-matching (winning) neuron $i(\mathbf{x})$ at time step n by using the minimum-distance Euclidean criterion:

$$i(\mathbf{x}) = \underset{j}{\operatorname{argmin}} \|\mathbf{x}(n) - \mathbf{w}_j\|, \quad j = 1,2,\ldots,J. \tag{1}$$

4. *Updating.* Adjust the synaptic weight vectors for all the neurons by using the updating formula:

$$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta(n) h_{j,i(\mathbf{x})}(n)(\mathbf{x}(n) - \mathbf{w}_j(n)), \tag{2}$$

where $\eta(n)$ is the learning rate parameter and $h_{j,i(\mathbf{x})}(n)$ is the neighborhood function centered around the winning neuron $i(\mathbf{x})$. Both $\eta(n)$ and $h_{j,i(\mathbf{x})}(n)$ are varied dynamically during this learning process to achieve the best results.

5. *Continuation.* Continue with step 2 until no noticeable changes in the feature map are observed.

The graphical representation obtained by the SOM approach only provides qualitative information, but quantitative information can be obtained by focusing on interesting regions of the map, although not the overall map. The overall map certainly has interesting properties, but more can be learned if the SOM (and thus the data) contains distinguishable regions that can be considered separately. Another option involves considering each map unit individually, but this is inconvenient for large maps. Thus, the efficient use of SOMs requires methods for identifying map unit clusters. It should be emphasized that the goal is not necessarily to optimize the clustering process, but instead the aim is to obtain insights into the cluster structure for the purpose of data mining. Therefore, the clustering method should be fast, robust, and visually efficient.

Vesanto and Alhoniemi (2000) proposed a two-level SOM involving K-means and the Davies–Bouldin (DB) index (Davies & Bouldin, 1979) for solving these problems (Fig. 2). This approach uses the SOM to determine the number of clusters and starting point, and then uses K-means to find the best cluster. In a single-level SOM, a large set of prototypes is formed initially instead of a data cluster. These prototypes can then be interpreted as proto-clusters, which are combined to form the actual clusters in the next phase. The benefit of using this SOM is that it can effectively reduce the computational time and noise, whereas higher abstraction levels yield greater distortion. The prototypes are less sensitive to random variations than the original data because the prototypes are local averages of the data (Vesanto & Alhoniemi, 2000).

The DB index is a function of the ratio of the total scatter within a cluster relative to the separation between the clusters. The best clustering of a two-level SOM is achieved by minimizing the following DB index, which is expressed as:

$$\frac{1}{K}\sum_{k=1}^{K} \max_{l \neq k} \left\{ \frac{S_c(Q_k) + S_c(Q_l)}{d_{ce}(Q_k, Q_l)} \right\}, \tag{3}$$

where S_c is the distance between nodes within a cluster, d_{ce} is the distance between clusters, and K is the number of clusters.

2.2. Reinforcement learning (RL) and Q-learning

RL addresses the problem of how an autonomous agent can learn to select appropriate actions to achieve its goals by interacting with its environment. In the RL framework, a learning agent must be able to perceive information in its environment in order to determine the current state of the environment. The agent then chooses an action to perform based on the perceived state. The action taken may result in a change in the state of the environment. Immediate reinforcement occurs based on the new state, which is used to reward or penalize the selected action. These interactions between the agent and its environment continue until the agent learns a decision-making strategy that maximizes the total reward. Stutton and Barto (1998) defined four key elements for addressing RL problems: a policy, reward function, value function, and model of the environment. The policy defines the agent's behavior for each of a number of given states. The reward function specifies the overall goal of the agent, which guides the agent while learning to achieve the goal. The value function specifies the value of a state or a state–action pair, thereby indicating its quality (the state or the state–action pair) in the long term. The model of the environment predicts the next state given the current state and a proposed action.

In addition to these four elements, a key assumption in the RL framework is that the definition of the current state used by each agent to make its decision should summarize everything important about the complete sequence of past states leading to it. Some of the information about the complete sequence may be lost, but it is only important that the future state should be contained within the current state signal. Therefore, the application environment should satisfy the Markovian property so the environment's next state can be predicted given the current state and action. Under this assumption, the interaction between an agent and its environment can be called a Markov decision process.

The original Q-learning algorithm was proposed by Watkins and Dayan (1992), 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 and action pair (denoted by s and a, respectively). It has been proved that the Q values learned using this algorithm converge to the optimal state–action values, Q^*. The optimal state–action values for a system represent the optimal policy that the agent aims to learn. The standard procedure for the Q-learning algorithm is given in Table 1.

Each iteration of steps 2–7 represents a learning cycle, which is also called an episode. The parameter α is the step size parameter and it influences the learning rate. The parameter γ is called the discount-rate parameter, $0 \leq \gamma \leq 1$, and it affects the present value of future rewards. The $Q(s, a)$ values can be initialized arbitrarily. If no actions are preferred for any specific states, then all the $Q(s, a)$ values in the policy table can be initialized with the same value when starting the Q-Learning procedure. If some prior knowledge of the benefit of certain actions is available, the agent may prefer taking these actions at the beginning by initializing the $Q(s, a)$ values as larger than the others. These actions will then be selected initially, which can shorten the learning period. Step 3 involves a tradeoff between exploration and exploitation, and many state–action pair selection methods may be used in this step.

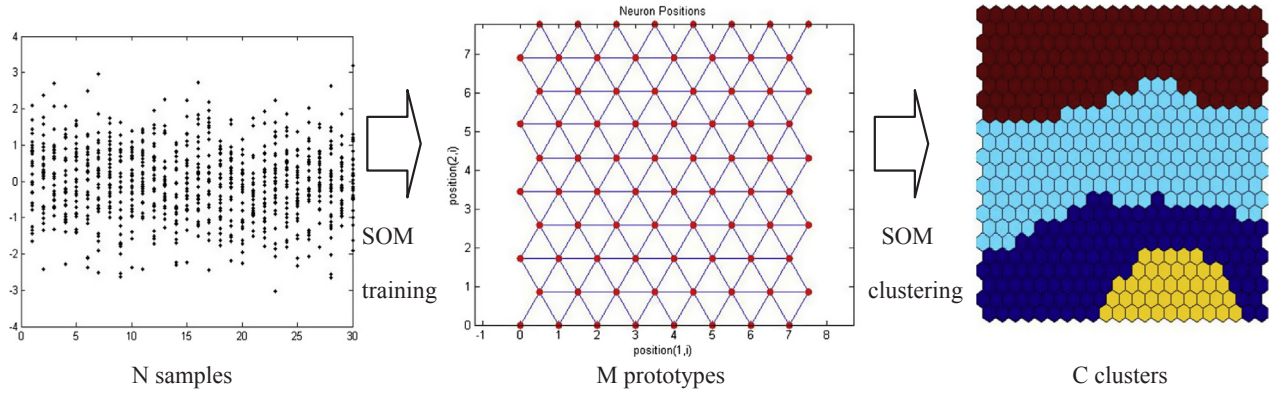

Fig. 2. A two-level SOM approach.

Table 1
Q-learning algorithm.

1. Initialize the $Q(s, a)$ value functions arbitrarily.
2. Perceive the current state, s
3. Following a certain policy (e.g. ε-greedy), select an appropriate action (a) for the given state (s)
4. Execute the selected action (a), receive immediate reward (r), and perceive the next state s'
5. Update the value function as follows:

$$Q(s,a) = Q(s,a) + \alpha[\gamma + \gamma \max_{a'} Q(s',a') - Q(s,a)] \quad (4)$$

6. Let $s \leftarrow s'$
7. Go to 3 until s state represents terminal state

3. Formulation of the problem

3.1. Real-time scheduling (RTS) using the MDRs mechanism

The status of a manufacturing system changes continuously, and previous studies have confirmed that it is possible to improve the system performance by implementing a multi-pass scheduling policy rather than using a single heuristic dispatching rule over an extended planning horizon. The policy is based on the system status at each decision point over a series of short-term scheduling period horizons. RTS based on machine learning has the advantage of rapidly yielding acceptable solutions for the operation of manufacturing systems.

RTS systems that use the MDRs mechanism can be represented by {**O**,**D**,**L**,**S**}, where S is the set of possible system states given by $\{s_1,s_2,...,s_T\}$ and it is described by a conjunction of system features f. At s_t, the MDRs for the following scheduling period ${}_o\mathbf{d}^{\text{next}}$ is implemented by the $l \in \mathbf{L}$ machine learning approach based on the current MDRs denoted as ${}_o\mathbf{d}^{\text{current}}$, the system features f, and the given performance criterion $o \in \mathbf{O}$. The vector ${}_o\mathbf{d}^{\text{next}} = [d_1,d_2,...,d_m]^{\text{T}}$, $d_i \in \mathbf{D}$ $i = 1,2,...,m$, D is the candidate dispatching rule used in RTS, and m is the number of machine cells on the shop floor. Therefore, the objective function for RTS using the MDRs mechanism can be defined as follows:

$$\mathbf{O}_l = \mathbf{O}_l^{\max} + \mathbf{O}_l^{\min} \quad (5)$$

where

$$\mathbf{O}_l^{\max} = \bigcup_o \sum_t \text{Max}\{f(\mathbf{f}, {}_o\mathbf{d}^{\text{current}}, l)\}$$

$$\mathbf{O}_l^{\min} = \bigcup_o \sum_t \text{Min}\{f(\mathbf{f}, {}_o\mathbf{d}^{\text{current}}, l)\}$$

and

$$\mathbf{O}_l^{\max} \cup \mathbf{O}_l^{\min} = \mathbf{O}_l, \text{ and } \mathbf{O}_l^{\max} \cap \mathbf{O}_l^{\min} = \varnothing.$$

3.2. Description of a study case

The study case involves a modification of the model used by Montazeri and Van Wassenhove (1990). This model is a true FMS involving: (i) three machine families (denoted as F1, F2, and F3), (ii) three load/unload stations, (iii) three automated guided vehicles, (iv) an input buffer and a central buffer with sufficient capacity to avoid deadlock, and (v) a computer-controlled local area network. The first two machine families have two machines each, whereas the third has a single machine. Fig. 1 shows the role of RTS using the MDRs mechanism in this study case. Eleven different types of parts are obtained by this model, and their routing and processing times are identical to those used by Montazeri and Van Wassenhove (1990).

3.3. Specification of the training examples

In the KB based on an RTS mechanism, let ${}_o\mathbf{X}$ denote the set of training examples and the given performance criterion o is obtained using the training sample generation mechanism. A training example ${}_o\mathbf{x} \in {}_o\mathbf{X}$ can be denoted by {f, ${}_o\mathbf{d}^{\text{current}}$, ${}_o\mathbf{d}^{\text{next}^*}$ }, where ${}_o\mathbf{d}^{\text{next}^*}$ are the acceptable (not necessarily optimal) MDRs for the following scheduling period under performance criterion o. Three types of performance criteria are typically studied in scheduling problems: throughput-based, flow-time-based, and due-date-based criteria. Table 2 lists the three performance criteria used in this study.

In this study, we consider the essential system attributes that satisfy various performance criteria. Therefore, all the possible system attributes are examined exhaustively. Table 3 lists 30 candidate attributes. The criteria for their selection are based on an earlier study (Arzi & Iaroslavitz, 2000; Chen, Yih, & Wu, 1999; Park, Raman, & Shaw, 1997; Su & Shiue, 2003), which considered machine learning and the features of the case study environment.

Dynamic dispatching rules are required because no optimal dispatching rule has yet been found for a variety of shop configurations and operating conditions (Baker, 1984; Blackstone, Philips, & Hogg, 1982; Montazeri and Van Wassenhove, 1990; Sabuncuoglu, 1998). Thus, excessive effort is not required to study the best dispatching heuristics in various environments. Table 4 summarizes five dispatching rules, which were shown to be effective in earlier studies (Montazeri and Van Wassenhove, 1990; Arzi & Iaroslavitz, 2000; Park

Table 2
Performance criteria used in this study.

Performance criteria	Description
TP	Throughput
MCT	Mean cycle time
NT	Number of tardy parts

Table 3
System attribute used in this study.

System attribute ID	Description
1	Number of the jobs in the system
2	The mean utilization of machines
3	The standard deviation of machine utilization
4	The mean utilization of load/unload stations
5	The mean utilization of pallet buffers
6	The mean utilization of AGVs
7	The minimum imminent operation time of candidate jobs within the system
8	The maximum imminent operation time of candidate jobs within the system
9	The mean imminent operation time of candidate jobs within the system
10	The standard deviation of the imminent operation time of candidate jobs within the system
11	The minimum total processing time of candidate jobs within the system
12	The maximum total processing time of candidate jobs within the system
13	The mean total processing time of candidate jobs within the system
14	The standard deviation of the total processing time of candidate jobs within the system
15	The minimum remaining processing time of candidate jobs within the system
16	The maximum remaining processing time of candidate jobs within the system
17	The mean remaining processing time of candidate jobs within the system
18	The standard deviation of the remaining processing time of candidate jobs within the system
19	The minimum slack time of candidate jobs within the system
20	The mean slack time of candidate jobs within the system
21	The standard deviation of the slack time of candidate jobs within the system
22	The maximum tardiness of candidate jobs within the system
23	The mean tardiness of candidate jobs within the system
24	The standard deviation of the tardiness of candidate jobs within the system
25	The maximum workload in front of any machine/station within the system
26	The total workload in front of any machine/station within the system
27	The mean sojourn time of candidate jobs within the system
28	The standard deviation of the sojourn time of candidate jobs within the system
29	The mean time now until due date of candidate jobs within the system
30	The standard deviation of the time now until due date of candidate jobs within the system

Table 4
Dispatching rules used in this study.

Dispatching rule	Description
DS	Select the job with minimum slack time
EDD	Select the job with the earliest due-date
SIO	Select the job with the shortest imminent operation time
SPT	Select the job with the shortest processing time
SRPT	Select the job with the shortest remaining processing time

et al., 1997) in terms of the three types of performance criteria mentioned above.

3.4. Research objective

As noted above, RTS plays an important role in the complex, dynamic, and highly stochastic operation of the shop floor environment. Hence, the proposed RTS using the MDRs mechanism must be capable of generating acceptable (if not optimal) MDRs in real time in order to enhance productivity by fully exploiting the dynamic nature of RTS, where this is essentially the concept of agile manufacturing. Therefore, the objective of this study is to develop an RTS using the MDRs strategy to greatly improve the production performance compared with classical SDR based on machine learning and by employing heuristic individual dispatching rules based on various performance criteria.

4. Development of RL-based RTS using the MDRs mechanism

The proposed RL-based RTS using the MDRs mechanism shown in Fig. 3 comprises two major modules. An off-line learning module runs a simulation to generate training examples to determine the system state number (built using a two-level SOM). Next, the system state number is sent to construct the initial state–action pair table for the RL module. During this operation, the Q-learning based agent is assumed to have the ability to perceive information from all the machines on the shop floor, and it is autonomously in charge of decision making for each individual machine. The aim of the decision-making process is to select an action (i.e., MDRs), where an action refers to the usage of a predefined weight vector for dispatching rules. The shop floor is considered as the environment for implementing RL. Basically, all the machines on the shop floor obey the agent's instructions. After all the machines complete an operation, a signal will be sent to the agent. The reward for the executed action is calculated by the agent and the respective state–action value is updated in the state–action value table. The Q-learning-based agent returns the instruction for the next action, which is selected according to a predefined policy. Finally, the Q-learning-based RL module updates the value function and determines the most appropriate MDRs for the next scheduling period. Each of these functions is discussed in detail in the following.

4.1. Simulation-based training example generation mechanism

A discrete event simulation model is used to generate the training examples to verify the proposed method. This approach is a modified version of that proposed by Arzi and Iaroslavitz (2000), and it involves a simulation based on some MDRs (using a random seed for each pass) for the same initial state of the system attributes and arriving job streams. In addition, in order to provide comprehensive training examples that represent a wide range of possible system states, the multi-pass simulation technique (Wu & Wysk, 1989) is used for collecting the training examples. The state variables of the system attributes are recorded as well as the MDRs at a decision point. At the end of the scheduling period, the MDRs are selected that yield the best performance measures as the action (i.e., decision) MDRs for a subset of the candidate MDRs (i.e., not including all of the MDRs) for the state recorded at the beginning of the scheduling period. Fig. 4 illustrates the multi-pass simulation technique used in this study for generating the training examples. The training example collection method has the advantage of being able to provide various job arrival patterns, which represent a wide range of possible system states, in order to learn the

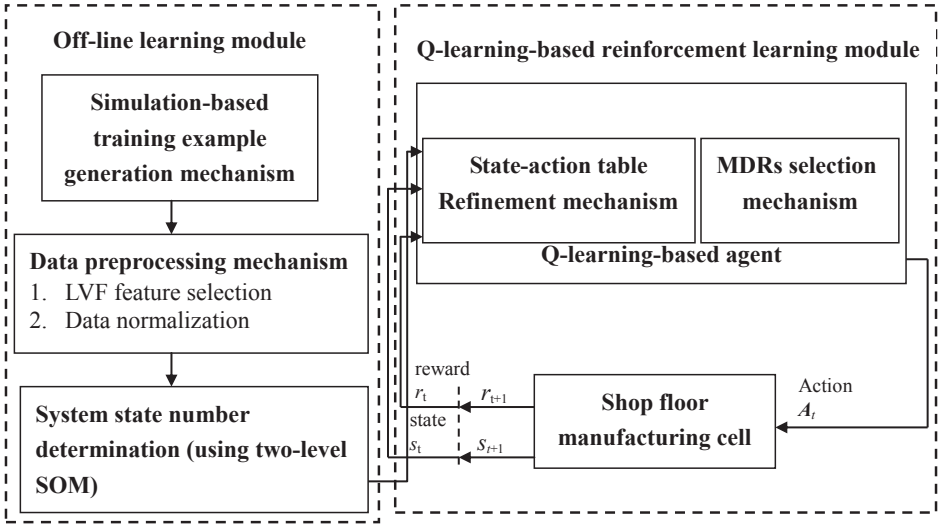

Fig. 3. The proposed RL-based RTS using the MDRs mechanism.

MDRs for the RTS KB in this study. Moreover, it can reduce the time required to generate the training examples.

4.2. Data preprocessing mechanism

In this study, the data preprocessing mechanisms comprise feature selection and data normalization, as described below.

4.2.1. Las vegas filter (LVF) feature selection

Previous studies (Chen et al., 1999; Su & Shiue, 2003) selected suitable system information based on various production requirements for constructing a RTS KB in machine learning-based RTS, and this is a crucial research issue because of the existence of a large amount of shop floor information in a manufacturing system. Using an excessive number of attributes may lead to overfitting of the training data and degrade the ability of the RTS system to generalize the KB. However, omitting even one important system attribute may strongly affect the learning process and the ability of the RTS system to classify knowledge.

Two general approaches are used for feature selection: the filter approach and wrapper approach (Liu & Motoda, 1998). In the filter approach, the selection process is conducted independently of the learning algorithm before applying the classifier to the selected feature subset. This is computationally more efficient than the wrapper approach, where the classifier system is given a feature subset as the input and the classification error is estimated using an unseen dataset. As mentioned above, RTS systems that use the MDRs mechanism cannot be developed with a machine learning algorithm using classifiers in order to construct the KB. Hence, we cannot apply the wrapper approach based on the predictive accuracy of a classifier for feature selection in our method.

The LVF feature selection algorithm is shown in Table 5 (Liu & Setiono, 1996). LVF generates a random feature subset (denoted as **RF**, where **RF** ⊂ **F**) from the feature subset space during each iteration. If **RF** contains fewer features than the current best subset, then the inconsistency rate of the dimensionally reduced data described by **RF** is compared with the inconsistency rate of the best subset. If **RF** is at least as consistent as the best subset, then **RF** replaces the best subset. The inconsistency rate of the training data prescribed by a given feature subset is defined over all groups of matching instances. Within a group of matching instances, the inconsistency count is the number of instances in the group minus the number of instances in the group with the most frequent class value. The overall inconsistency rate is the sum of the inconsistency counts for all groups of matching instances divided by the total number of instances.

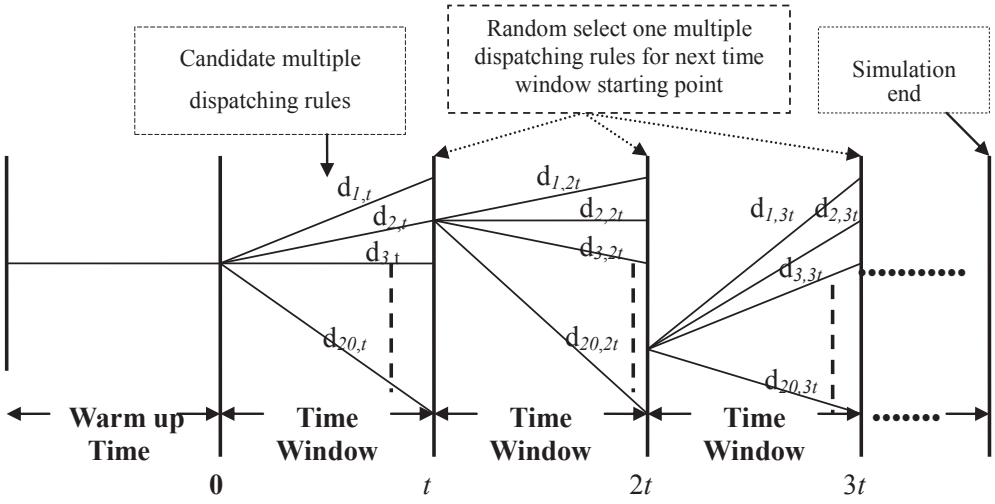

Fig. 4. The multi-pass technique to generate the training samples procedure.

Table 5
LVF algorithm.

```
LVF algorithm.Input: Max-TRIES,
        D-dataset,
        n-number of features,
        r-allowable inconsistency rate;
initialize: C_best ← n;
for Max-TRIES loops
begin
  RF ← randomFeatureSet (seed);
  C ← numberOfFeatures (RF);
  if (C < C_best)
    if (InconCheck (RF,D) < r;
      RF_best ← RF; C_best ← C;
      Print_Current_Best (RF)
    Else if ((C_best = C) and (InconCheck (RF,D) < r))
      Print_ Current_Best (RF)
end
Output: sets of m features satisfying the inconsistency criterion
```

4.2.2. Data normalization

A feature is normalized by scaling its values so they fall within a small specified range, such as −1.0 to +1.0. Normalization is particularly appealing with distance-based classifiers because after normalizing the attribute values, we can avoid attributes with large values, which would dominate those with smaller values. The three widely used normalization methods are min-max normalization, z-scores normalization, and normalization by decimal scaling (Han & Kamber, 2006).

Min-max normalization is most suitable for cases where the upper and lower bounds of the scores are known. The minimum and maximum scores are mapped to 0 and 1, respectively. The z-scores normalization method is useful when the actual minimum and maximum values of a feature are unknown, or when outliers dominate the min-max normalization. Decimal scaling can be applied when the values of different training examples are distributed on a logarithmic scale. For example, if one training example has values in the range of [0, 1] and another is in the range of [0, 1000], then decimal scaling normalization can be applied. The values of the system attributes change continually on the shop floor. Hence, the z-scores normalization method is suitable for attribute data normalization in this study.

During z-scores normalization, the values for a feature f are normalized using the mean and standard deviation of f. A value v of f is normalized to v' by the following equation:

$$v' = \frac{v - \overline{f}}{\sigma_f}, \tag{6}$$

where $\overline{f}$ and σ_f are the mean and standard deviation, respectively, of feature f.

4.3. Criterion for system state number determination

In RL, the agent perceives the current system states in the environment and then selects the most appropriate action to be performed. In the classical machine RTS approach, the number of system states is infinite. However, state–action tables are constructed to represent the knowledge that the RL agent perceives by investigating the results of actions taken in each states, so infinite system states are not feasible for an RL agent.

Thus, the training examples obtained by LVF feature selection under performance criterion o are denoted as ${}_o\mathbf{X}_{\mathrm{LVF}}$ (i.e., ${}_o\mathbf{X}_{\mathrm{LVF}}$ is the tuple $\{\mathbf{f}_{\mathrm{LVF}}, {}_o\mathbf{d}^{\mathrm{current}}, {}_o\mathbf{d}^{\mathrm{next}^*}\}$), where the system features obtained by LVF feature selection for the training examples are denoted as $\mathbf{f}_{\mathrm{LVF}}$. By using two-level SOM clustering analysis, ${}_o\mathbf{X}_{\mathrm{LVF}}$ can be classified into various system state class. If the training data are in the same class, they can be assigned the same system state number labels. A training example with the same system state number label i ($i = 1, \ldots\ldots k$) denoted as ${}_o\mathbf{x}^i_{\mathrm{LVF}} \in {}_o\mathbf{X}^i_{\mathrm{LVF}}$ becomes a four-tuple $\{\mathbf{f}_{\mathrm{LVF}}, {}_o\mathbf{d}^{\mathrm{current}}, {}_os^i\ {}_o\mathbf{d}^{\mathrm{next}^*}\}$, where ${}_o\mathbf{X}^i_{\mathrm{LVF}}$ represents the subset of i (i.e., assign class label i) in ${}_o\mathbf{X}_{\mathrm{LVF}}$ obtained by clustering analysis under performance criterion o, and ${}_os^i$ is the system state number label i. Fig. 5 shows the training examples for the two-level SOM process. The specific system state class number is assigned in each training example.

4.4. Q-learning-based agent

4.4.1. State–action table refinement mechanism

For the Q-learning-based agent, state–action tables are constructed to represent the knowledge that the Q-learning-based agent learns by investigating the results of actions taken in every state. In this study, an action $a_{j^{(i)}}$ is defined as the Q-learning-based agent's choice weight vector ${}_o\mathbf{d}^{\mathrm{next}}$ based on perceived state ${}_os^i$ of the shop floor, where $a_{j^{(i)}}$ is the ith state class in the j-th ($j = 1, 2, \ldots, n_i$) training example's (i.e., $\mathbf{x}_{n_j^{(i)}}$) action. Each state–action pair is associate with a Q value: Q (${}_os^i$, $a_{j^{(i)}}$). A sample of a state–action table for the TP performance criterion is shown in Table 6.

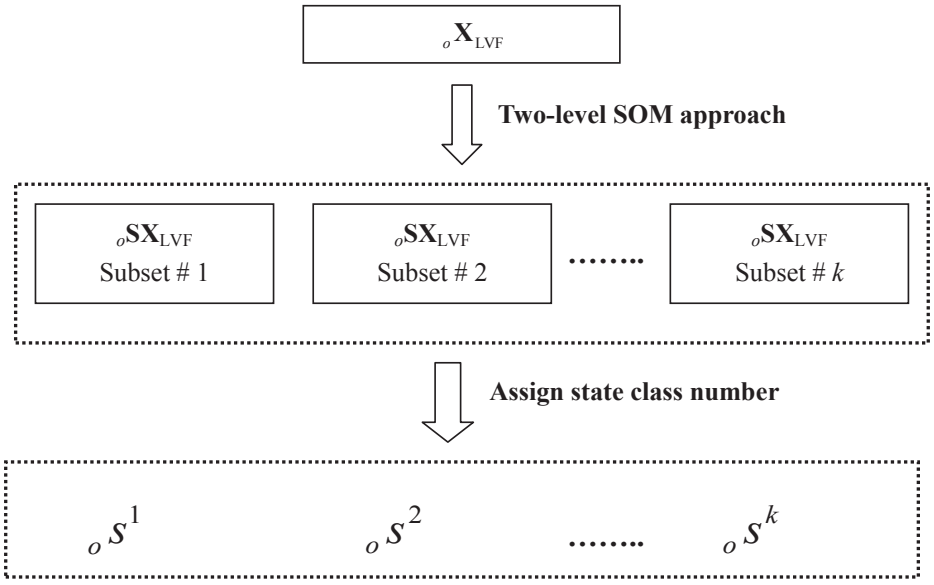

Fig. 5. Assignment of specific system state class number using two-level SOM in training examples.

Table 6
State–action table for the TP performance criterion.

State class number	Training sample number	Q value
${}_{TP}s^1$	$\mathbf{x}_{1(1)}$	$Q({}_{TP}s^1, a_{1(1)})$
${}_{TP}s^1$	$\mathbf{x}_{2(1)}$	$Q({}_{TP}s^1, a_{2(1)})$
…	…	…
${}_{TP}s^1$	$\mathbf{x}_{n_1^{(1)}}$	$Q({}_{TP}s^1, a_{n_1^{(1)}})$
…	…	…
${}_{TP}s^k$	$\mathbf{x}_{1(k)}$	$Q({}_{TP}s^k, a_{1(k)})$
${}_{TP}s^k$	$\mathbf{x}_{2(k)}$	$Q({}_{TP}s^k, a_{2(k)})$
…	…	…
${}_{TP}s^k$	$\mathbf{x}_{n_k^{(k)}}$	$Q({}_{TP}s^k, a_{n_k^{(k)}})$

The initial determination of the state–action value $Q\ ({}_os^i,\ a_{j(i)})$ is based on the performance measures for a set of training examples and the given performance criterion o $({}_o\mathbf{X})$, where it is calculated by Eq. (7), and the Q value for a state–action pair is updated by the Q-learning-based agent using Eq. (11).

$$Q({}_os^i, a_{j(i)}) = \left(\frac{{}_oPM_{j(i)} - {}_oPM^i_{min}}{{}_oPM^i_{max} - {}_oPM^i_{min}}\right), \quad o \in \{TP\}$$
$$Q({}_os^i, a_{j(i)}) = 1 - \left(\frac{{}_oPM_{j(i)} - {}_oPM^i_{min}}{{}_oPM^i_{max} - {}_oPM^i_{min}}\right), \quad o \in \{MCT, NT\} \tag{7}$$

where ${}_oPM_{j(i)}$ is training sample $\mathbf{x}_{n_j^{(i)}}$ at the scheduling decision point for the performance measure under performance criterion o, ${}_oPM^i_{min}$ is the minimum value of the performance measure in state class number i, and ${}_oPM^i_{max}$ is the maximum value of the performance measure in state class number i.

4.4.2. MDRs selection mechanism

The following problem must be addressed after establishing a state–action table refinement mechanism. A specific system state class number can be determined by using the two-level SOM, but it is not clear how to choose the most appropriate MDRs for the next scheduling period among the many candidate MDRs in a KB class label. Thus, we propose a method for selecting MDRs to solve this problem according to two types of key strategies: the policy for selecting an action (i.e., MDRs) and the reward function in MDRs selection mechanism must conform with the RL framework concept.

In RL, MDRs are selected via the exploration and exploitation of two types of strategies for choosing an action. Exploration requires that the agent tries something different in order to obtain a greater reward, whereas the agent favors actions taken previously and rewarded during exploitation. Exploitation may have the advantage of guaranteeing a good expected reward in one play, whereas exploration provides more opportunities for finding the maximum total reward in the long term. A popular approach for addressing this trade-off issue is the ε-greedy policy. The ε-greedy policy involves selecting the action with the best value (exploitation) with probability 1–ε; otherwise, an action is selected randomly with a small probability ε.

After the agent has been trained, all state–action pair values have been updated and each state has a dominated action. The greedy policy is then used to select an action, which is defined as follows.

$$\pi_t({}_os^i, a_{j(i)}) = \underset{a_{j(i)}}{\operatorname{argmax}}\ ({}_os^i, a_{j(i)}) \tag{8}$$

The goal of the RL module is to learn the best choice from the MDRs at each decision point, where the agent learns by interacting directly with the system and responding to the receipt of rewards or penalties defined by a reward function, which is based on the impact each action has on the system. The reward function defines the goal for the learning agent and determines the value of the immediate action based on the perceived state of the environment. The learning agent tries to maximize the total reward, so the reward function is essentially used to guide the learning agent toward its goal.

Table 7
Reward function.

TP criterion		MCT and NT criteria
if ${}_{TP}PM^i_{SP} > {}_{TP}UCL^i_{1\sigma}$		**if** ${}_{MCT}PM^i_{SP} > {}_{MCT}UCL^i_{1\sigma}$ or ${}_{NT}PM^i_{SP} > {}_{NT}UCL^i_{1\sigma}$
Reward = +1		Reward = −1
else if ${}_{TP}PM^i_{SP} < {}_{TP}LCL^i_{1\sigma}$	(9)	**else if** ${}_{MCT}PM^i_{SP} < {}_{MCT}UCL^i_{1\sigma}$ or ${}_{NT}PM^i_{SP} < {}_{NT}UCL^i_{1\sigma}$ (10)
Reward = −1		Reward = +1
else		**else**
Reward = 0		Reward = 0
end if		**end if**

In the study case based on the TP criterion, the system's objective is to maximum the throughput. After the one-step scheduling period is finished, the online simulation output is compared with the mean performance in state class label i. If the performance measure for the one-step scheduling period output denoted as ${}_{TP}PM^i_{SP}$ is greater than the one sigma upper confidence limit (i.e., 68.27%) for the mean performance in state class label i denoted as ${}_{TP}UCL^i_{1\sigma}$, then the learning agent receives a reward of +1. In addition, if ${}_{TP}PM^i_{SP}$ is less than the one sigma lower confidence limit (i.e., 31.73%) for the mean performance in state class label i denoted as ${}_{TP}LCL^i_{1\sigma}$, then the learning agent receives a reward of −1. Otherwise, the learning agent receives a reward of 0. The detailed reward function employed in this study is shown in Table 7.

The overall proposed Q-learning-based agent operating procedure is shown in Table 8. The state–action table associated with the Q value is update and the most appropriate MDRs are determined for the next scheduling period.

5. Experiment

5.1. Construction of a simulation model and generation of a training example

To verify the proposed method, a discrete event simulation model was used to generate training examples. The simulation model was built and executed using Tecnomatix Plant Simulation (2006), an object-oriented simulation language, and it was run on a Core i7-4790 3.6 GHz CPU with the Windows 7 operating system.

It was expected that the proposed approach would achieve the desired dynamic dispatching performance. Several parameters were determined based on a preliminary simulation run. The time between jobs followed an exponential distribution with a mean of 31 min. The due date for each job was randomly assigned from six to 10 times the total

Table 8
Proposed Q-learning based agent operating procedure.

Step 1 **Input:** performance criterion o, current system status and MDRs
Step 2 **initialize**: $Q\ ({}_os^i, a_{j(i)})$ // calculate by Eq. (7) **and** $\mathbf{d}_o^{next^*} \varnothing$;
Step 3 Determine matching state class no. i by two-level SOM approach
Step 4 Select an appropriate MDRs$\mathbf{d}_o^{next^*}$ (i.e., action $a_{j(i)}$) for next scheduling interval by ε -greedy policy.
Step 5 Execute the selected action $(a_{j(i)})$, receive immediate reward (r) by Eq. (9) or (10), and perceive the next state s'
Step 6 Update the Q value of state-action pair as follows:
$$Q(s,a_{j(i)}) = Q(s,a_{j(i)}) + \alpha[r + \gamma \max_{a'_{j(i)}} Q(s',a'_{j(i)}) - Q(s,a_{j(i)})] \tag{11}$$
Step 7 $s \leftarrow s'$
Step 8 **Go Back to Step 3** until s state represents terminal state.

Table 9
Five part mix ratios generate training examples and training examples KB.

Part ID	Part mix ratios (%)				
	Mix 1	Mix 2	Mix 3	Mix 4	Mix 5
1	11.00	14.00	6.00	9.00	14.00
2	11.00	14.00	6.00	9.00	14.00
3	11.00	15.00	6.00	9.00	14.00
4	12.00	10.00	15.00	8.00	15.00
5	6.00	12.00	15.00	13.00	7.00
6	8.00	8.00	9.00	12.00	5.00
7	8.00	5.00	8.00	3.00	5.00
8	7.00	3.00	8.00	9.00	4.00
9	7.00	3.00	7.00	8.00	4.00
10	2.50	1.00	4.00	1.00	6.00
11	16.50	15.00	16.00	19.00	12.00

Table 10
The result of selected system feature for each performance criterion.

Performance criterion	Selected system feature ID	No. of feature selected
TP	{1, 9, 13, 17, 26}	5
MCT	{1, 2, 17, 18, 26}	5
NT	{1, 2, 10, 13, 18}	5

Table 11
SOM parameters used in this study.

Map size	10 × 10
Lattice	Hexagonal
Shape	Sheet
Neighborhood function	Gaussian

processing time and it was uniformly distributed. The maximum number of pallets (jobs) permitted in the FMS system was 100. Table 9 shows the five product-mix ratios used to generate the training examples. The proportions of part types varied continually every 20,000 min.

In order to generate a large number of different training examples, we used 100 different random seeds to generate 100 different job arrival patterns. The warm-up period for each run was 10,000 min and it was followed by 60 multi-pass scheduling periods. The time window for the multi-pass simulation was 2000 min. In total, 6000 training samples were collected

5.2. Development of the off-line learning module

According to Fig. 3, it was necessary to choose the system features to construct the RL system state class number (build using the two-level SOM) in advance. The LVF feature selection algorithm was encoded in MATLAB 7.1 (MathWorks, 2005), where the inconsistency rate was set to 0.5. Table 10 shows the results obtained by LVF feature selection.

In this study, the two-level SOM was used for clustering the training samples and it assigned a system state class number label to each performance criterion. The two-level SOM algorithm was encoded using the MATLAB Neural Network Toolbox (MathWorks, 2007). Some

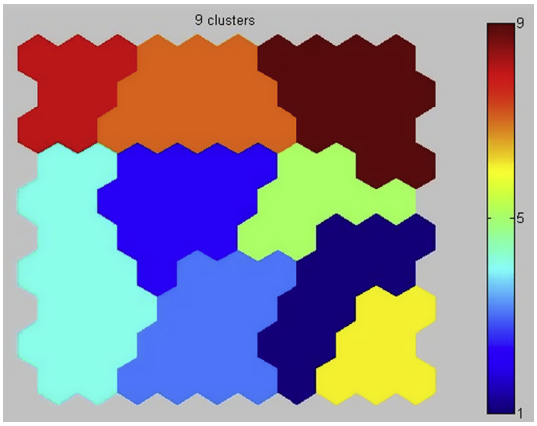

Fig. 6. Shows the best clustering SOM U-matrix in TP criterion.

Table 13
Results obtained using the RL-based MDRs approach and other scheduling strategies according to three production criteria.

Scheduling strategy	TP		MCT (minutes)		NT	
	Mean	SD	Mean	SD	Mean	SD
RL	13122.66	33.47	947.98	206.81	1442.80	338.22
SOM	13085.57	25.64	1155.95	270.37	1450.13	499.89
GA + DT	13067.90	40.71	1327.98	314.22	1601.77	649.02
GA + SVM	13078.56	37.51	1417.50	273.40	1588.77	668.74
DS	13048.60	78.36	1569.73	395.94	3189.90	1855.20
EDD	12978.17	91.22	2065.82	437.31	8021.40	2226.72
SPT	13061.00	49.00	1440.84	297.91	1759.10	713.56
SIO	13017.37	80.55	1673.51	336.97	3076.73	1575.75
SRPT	13056.77	43.30	1536.91	346.79	1630.60	750.74

SD, standard deviation.

experimental parameters used in the two-level SOM are shown in Table 11. Table 12 shows the DB index, which used a K-mean value (i.e., the number of clusters) that ranged from 2 to 10 for each performance criterion. Thus, the minimum DB indexes in TP, MCT, and NT were nine, 10, and six, respectively. Fig. 6 shows the best clustering SOM U-matrix (unified distance matrix) obtained for the TP performance criterion. According to the results in Table 12, in order to meet the TP, MCT, and NT performance criteria, we required 9, 10, and 6 system state class number labels, respectively. Next, the system state class number for each performance criterion was established as described in Section 4.3.

5.3. Experimental verification by online simulation

A Q-learning-based agent encoded using the Tecnomatix Plant SimTalk simulation object-oriented programming language (2006) was linked to a C program (by employing a two-level SOM coded using MATLAB for determining the system state class number) to examine the effectiveness of the proposed RL-based RTS using the MDRs mechanism in various system scenarios.

A stream of arriving jobs was generated in a simulation over

Table 12
DB index of each performance criterion.

Performance criterion	No. of cluster								
	2	3	4	5	6	7	8	9	10
TP	0.8935	1.0368	0.9520	0.9548	0.7694	0.8258	0.8415	**0.7597**	0.8147
MCT	0.8359	0.9965	0.8714	0.9630	0.8828	0.8664	0.8415	0.8383	**0.8347**
NT	0.9765	1.1976	0.9756	0.9034	**0.8532**	1.2367	1.0395	0.8765	0.9266

Table 14
Results of the paired-sample *t*-tests to compare the performance of the RL-based MDRs approach and other scheduling strategies according to three production criteria.

Performance criterion	*P*-value							
	SOM	GA + DT	GA + SVM	DS	EDD	SPT	SIO	SRPT
TP	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
MCT	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
NT	0.034	0.000	0.000	0.000	0.000	0.000	0.000	0.000

400,000 min by using different sets of random seeds to investigate whether the proposed RL-based RTS using the MDRs mechanism was more effective over a long period than the RTS using the previously proposed SOM-based MDRs method (Guh et al., 2011), as well as SDR approaches based on machine learning, i.e., GA + DT (Su & Shiue, 2003) and GA + SVM (Shiue, 2009). In addition, a single heuristic dispatching rule was compared in various scenarios based on three performance criteria. Table 13 shows the mean and standard deviation based on 30 simulation runs using 30 random seeds with different scheduling strategies. The proposed approach was capable of obtaining better results according to all the performance criteria because of its superior effectiveness.

A paired-samples *t*-test was used to determine whether the proposed RL-based MDRs approach was significantly better than the SOM-based MDRs, the GA + DT and GA + SVM approaches, and heuristic dispatching strategies. The null hypothesis was that the mean values obtained by all the scheduling strategies were equal. As shown in Table 14, the null hypothesis was rejected at a significance level of 95% for all the control strategies. Therefore, the proposed RL-based MDRs approach was significantly better than the SOM-based MDRs, SDR approaches, and other dispatching strategies.

6. Conclusion and future work

In this study, we proposed an RL-based MDRs selection mechanism for constructing an RTS for FMS control. We provide the following conclusions based on the results of this study.

- The proposed RL-based approach using the MDRs selection mechanism responds efficiently to changes in the shop floor environment and it is suitable for incorporating in the operation of an RTS system for a smart factory.
- The proposed RL-based approach employs an intelligent and dynamic method for selecting MDRs, which is based on the status of a manufacturing system at the end of a given scheduling period, where it then determines the appropriate MDRs for the following period. Our results showed that over a long period, according to various performance criteria, this approach performed better than a previously proposed SOM-based MDRs selection mechanism, the machine learning-based RTS using the SDR approach, and heuristic individual dispatching rules.

Some potential issues for future research can be identified based on the results of this study, as follows.

- The FMS model used for the experimental verification in this study is a relatively simple manufacturing system. Thus, it is necessary to apply the proposed approach to large and complex manufacturing systems, such as semiconductor wafer fabrication systems. The effectiveness of the RL-based MDRs selection mechanism might differ in more complex semiconductor wafer fabrication systems that incorporate input-order wafer lot release control and the selection of a wafer lot via an intrabay by a stocker. Investigating and examining proposed RL-based approach using the MDRs selection mechanism feasibility through various manufacturing systems would be a potential topic for future research.
- Recently, deep learning, which is a type of machine learning approach, has attracted much attention in academic and commercial research (Bengio, 2009), where it has been applied successfully to classification tasks, automatic speech recognition, image recognition, and self-driving cars (LeCun, Bengio, & Hinton, 2015). Deep learning algorithms use a deep architecture to extract the inherent features of data from the lowest level to the highest level, and they can discover large amounts of structure in data. Due to the existence of a large amount of shop floor information in CPS, deep learning algorithms can represent CPS features without prior knowledge. These characteristics may inspire future studies of RTS for smart factories by employing deep learning algorithms.

References

Arzi, Y., & Iaroslavitz, L. (2000). Operating an FMC by a decision-tree-based adaptive production control system. *International Journal of Production Research, 38*(3), 675–697.

Baker, K. R. (1984). Sequencing rules and due-date assignments in a job shop. *Management Science, 30*(9), 1093–1104.

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends® in Machine Learning, 2*(1), 1–127.

Blackstone, J. H., Philips, D. T., Jr., & Hogg, G. L. (1982). A state-of-the-art survey of dispatching rules for manufacturing job shop operations. *International Journal of Production Research, 20*(1), 27–45.

Chen, C. C., Yih, Y., & Wu, Y. C. (1999). Auto-bias selection for learning-based scheduling systems. *International Journal of Production Research, 37*(9), 1987–2002.

Davies, F. D., & Bouldin, D. W. (1979). A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 1*(2), 224–227.

Goryachev, A., Kozhevnikov, S., Kolbova, E., Kuznetsov, O., Simonova, E., Skobelev, P., ... Shepilov, Y. (2013). Smart factory: Intelligent system for workshop resource allocation, scheduling, optimization and controlling in real time. *Advanced Materials Research, 630*, 508–513.

Guh, R. S., Shiue, Y. R., & Tseng, T. Y. (2011). The study of real time scheduling by an intelligent multi-controller approach. *International Journal of Production Research, 49*(10), 2977–2997.

Han, J., & Kamber, M. (2006). *Data mining concepts and techniques* (2nd ed.). San Francisco, CA: Morgan Kaufmann.

Herrmann, M., Pentek, T., & Otto, B. (2016). Design principles for industrie 4.0 scenarios. In: *49th Hawaii international conference on system sciences, Koloa, HI, USA* (pp. 3928–3937).

Ishii, N., & Talavage, J. J. (1994). A mixed dispatching rule approach in FMS scheduling. *International Journal of Flexible Manufacturing Systems, 6*(1), 69–87.

Kim, C. O., Min, H. S., & Yih, Y. (1998). Integration of inductive learning and neural networks for multi-objective FMS scheduling. *International Journal of Production Research, 36*(9), 2497–2509.

Kohonen, T. (2001). *Self-organizing map* (3rd ed.). Verlag, NY: Springer.

Kück, M., Ehm, J., Freitag, M., Frazzon, E. M., & Pimentel, R. (2016). A data-driven simulation-based optimisation approach for adaptive scheduling and control of dynamic manufacturing systems. *Advanced Materials Research, 1140*, 449–456.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521*(7553), 436–444.

Lee, E. A. (2008). Cyber physical systems: Design challenges. In: *11th IEEE symposium on object oriented real-time distributed computing, Orlando, FL, USA* (pp. 363–389).

Lee, J., Bagheri, B., & Kao, H. A. (2015). A cyber-physical systems architecture for industry 4.0-based manufacturing systems. *Manufacturing Letters, 3*, 18–23.

Lee, J., Kao, H. A., & Yang, S. (2014). Service innovation and smart analytics for industry 4.0 and big data environment. *Procedia CIRP, 16*, 3–8.

Lee, J., Lapira, E., Bagheri, B., & Kao, H. (2013). Recent advances and trends in predictive manufacturing systems in big data environment. *Manufacturing Letters, 1*, 38–41.

Liu, H., & Setiono, R. (1996). A probabilistic approach to feature selection—a filter solution. In: *Proceedings of the 13th international conference on machine learning, Bari, Italy* (pp. 319–327).

Liu, H., & Motoda, H. (1998). *Feature selection for knowledge discovery and data mining.* Boston, MA: Kluwer Academic Publishers.

MathWorks (2005). *Matlab version 7.1 user's guide.* Natick, MA: MathWorks Inc.

MathWorks (2007). *Matlab neural network toolbox version 5 user's guide.* Natick, MA:

MathWorks Inc.
Metan, G., Sabuncuoglu, I., & Pierreval, H. (2010). Real time selection of scheduling rules and knowledge extraction via dynamically controlled data mining. *International Journal of Production Research, 48*(23), 6909–6938.
Montazeri, M., & Van Wassenhove, L. N. (1990). Analysis of scheduling rules for an FMS. *International Journal of Production Research, 28*(4), 785–802.
Olafsson, S., & Li, X. (2010). Learning effective new single machine dispatching rules from optimal scheduling data. *International Journal of Production Economics, 128*(1), 118–126.
Park, S. C., Raman, N., & Shaw, M. J. (1997). Adaptive scheduling in dynamic flexible manufacturing systems: A dynamic rule selection approach. *IEEE Transactions on Robotics and Automation, 13*(4), 486–502.
Priore, P., Gómez, A., Pino, R., & Rosillo, R. (2014). Dynamic scheduling of manufacturing systems using machine learning: An updated review. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 28*(1), 83–97.
Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Mateo, CA: Morgan Kaufman.
Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). *Learning internal representations by error propagation. Parallel distributed processing.* Cambridge, MA: MIT Press.
Sabuncuoglu, I. (1998). A study of scheduling rules of flexible manufacturing systems: A simulation approach. *International Journal Production Research, 36*(2), 527–546.
Shahrabi, J., Adibi, M. A., & Mahootchi, M. (2017). A reinforcement learning approach to parameter estimation in dynamic job shop scheduling. *Computers & Industrial Engineering, 110*, 75–82.
Shiue, Y. R. (2009). Data mining-based dynamic dispatching rule selection mechanism for shop floor control systems using support vector machine approach. *International Journal of Production Research, 47*(13), 3669–3690.
Shiue, Y. R., Guh, R. S., & Lee, K. C. (2012). Development of machine learning-based real time scheduling systems: Using ensemble based on wrapper feature selection approach. *International Journal of Production Research, 50*(20), 5887–5905.
Shiue, Y. R., Guh, R. S., & Tseng, T. Y. (2012). Study on shop floor control system in semiconductor fabrication by self-organizing map-based intelligent multi-controller. *Computer & Industrial Engineering, 62*(4), 1119–1129.
Son, Y., Rodriguez-Rivera, H., & Wysk, R. A. (1999). A multi-pass simulation-based real-time scheduling and shop floor control system. *Transactions of the Society for Computer Simulation International, 16*(4), 159–172.
Stutton, R. S., & Barto, G. B. (1998). *Reinforcement learning: An introduction.* Cambridge, MA: MIT Press.
Su, C. T., & Shiue, Y. R. (2003). Intelligent scheduling controller for shop floor control systems: A hybrid genetic algorithm/decision tree learning approach. *International Journal of Production Research, 41*(12), 2619–2641.
Tecnomatix Plant Simulation (2006). *Tecnomatix plant simulation 7.6 user guide*. Tecnomatix Technologies Ltd: Plano.
Vapnik, V. N. (2000). *The nature of statistical learning theory* (2nd ed.). New York, NY: Springer.
Vesanto, J., & Alhoniemi, E. (2000). Clustering of the self-organizing map. *IEEE Transactions on Neural Networks, 11*(3), 586–600.
Wang, Y. C., & Usher, J. M. (2005). Application of reinforcement learning for agent-based production scheduling. *Engineering Applications of Artificial Intelligence, 18*(1), 73–82.
Wang, S., Wan, J., Li, D., & Zhang, C. (2016). Implementing smart factory of industrie 4.0: An outlook. *International Journal of Distributed Sensor Networks, 12*(1), http://dx.doi.org/10.1155/2016/3159805 Art. ID 3159805.
Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning, 8*(3), 279–292.
Wolf, W. (2009). Cyber-physical systems. *IEEE Computer, 42*(3), 88–89.
Wu, S. D., & Wysk, R. A. (1989). An application of discrete-event simulation to online control and scheduling in flexible manufacturing. *International Journal of Production Research, 27*(9), 1603–1623.