Available online at www.sciencedirect.com
SCIENCE DIRECT®
Engineering Applications of Artificial Intelligence 18 (2005) 73–82
Engineering Applications of
ARTIFICIAL INTELLIGENCE
www.elsevier.com/locate/engappai

Application of reinforcement learning for agent-based production scheduling

Yi-Chi Wang[1], John M. Usher*

Department of Industrial Engineering, Mississippi State University, 260 McCain Bldg., P.O. Box 9542, Miss. State, MS39762, USA

Received 19 August 2003; accepted 16 August 2004
Available online 25 September 2004

Abstract

Reinforcement learning (RL) has received some attention in recent years from agent-based researchers because it deals with the problem of how an autonomous agent can learn to select proper actions for achieving its goals through interacting with its environment. Although there have been several successful examples demonstrating the usefulness of RL, its application to manufacturing systems has not been fully explored yet. In this paper, *Q*-learning, a popular RL algorithm, is applied to a single machine dispatching rule selection problem. This paper investigates the application potential of *Q*-learning, a widely used RL algorithm to a dispatching rule selection problem on a single machine to determine if it can be used to enable a single machine agent to learn commonly accepted dispatching rules for three example cases in which the best dispatching rules have been previously defined. This study provided encouraging results that show the potential of RL for application to agent-based production scheduling.

Keywords: Reinforcement learning; *Q*-learning; Dispatching rule selection

1. Introduction

1.1. Manufacturing scheduling

A long and profitable life is every enterprise's goal. For a manufacturing enterprise, to maintain profitability requires that they continually excel in converting raw materials into value-added products that meet the customers' needs. This conversion procedure consists of a set of complicated and interrelated activities such as designing, planning, production, inventory control, quality assurance, etc. To remain competitive in the market, manufacturers must focus on continually improving their processes. Production scheduling that translates the detailed process plans into the shop floor schedule is one of the most important processes in manufacturing systems. A good production schedule can provide such benefits as increased shop throughput, enhanced customer satisfaction, lower inventory levels, and increased utilization of resources. Therefore, there is a great need for good scheduling strategies.

Scheduling problems essentially involve completing a set of jobs with a limited number of manufacturing resources under a number of constraints to optimize a particular objective function. These problems are known to be hard and usually belong to the NP-complete class of problems (Morton and Pentico, 1993; Pinedo, 1995). Research in production scheduling has been conducted for many decades and a large number of algorithms and heuristics have been developed for various scheduling problems. A scheduling problem consists of three components: a machine environment, specific job characteristics, and one or more optimality criterion (Brucker, 2001). The machine environment represents

*Corresponding author. Tel.: +1-662-325-7624; fax: +1-662-325-7618.
E-mail address: usher@ie.msstate.edu (J.M. Usher).
[1]The author is currently with Department of Information Management, Kun Shan University of Technology, No 949, Da Wan Rd., Yung-Kang City, Tainan Hsien, 710, Taiwan.

0952-1976/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.engappai.2004.08.018

the type of the manufacturing system that will execute the developed schedule. The manufacturing system may be a job shop system, flexible manufacturing system (FMS), cellular manufacturing system, transfer line, etc. Job characteristics represent such factors as the number of operations, the precedence relations among operations, and the possibility of preemption (whether the job can be split). Optimality criteria are the objectives to pursue when scheduling the jobs. Common objectives include minimizing makespan, mean flow time, mean lateness, the number of tardy jobs, and mean tardiness. All the three components mentioned above specify the variety and complexity of each scheduling problem.

A scheduling problem may be comprised of two subproblems: job routing and job sequencing problems. A job routing problem involves assigning the operations of jobs to the specific machines. Such problems result from the allowance of routing flexibility. Routing flexibility depends on the capability of the machines. A versatile machine is capable of performing different operations. The versatility of the various machines in a shop essentially supports the possibility for the existence of alternative process plans for a job. Routing flexibility is a key issue that has increasingly attracted attention in modern manufacturing systems. A FMS, which consists of a set of computer numerically controlled machines (CNC) linked with an automated material handling system, is a computerized system that is able to produce mid-volume and mid-variety products with high levels of efficiency. The FMS provides routing flexibility due to the capability of NC machines.

Once the route of a job is specified, decision makers must determine the production sequence of the jobs awaiting their next process in the machine queue. That is a job sequencing problem. A simple approach to such problems is to adopt dispatching rules. A dispatching rule is a priority rule used to determine the order in which the jobs waiting in the machine queue are to be processed as soon as a machine becomes available. Dispatching rules are useful for finding a reasonably good schedule. The dispatching rules are attractive because of their simplicity and ease of implementation. A variety of dispatching rules have been proposed in recent decades, with Panwalkar and Iskander (1977) identifying the existence of more than 100 distinct rules. Scheduling in industry may require meeting several objectives simultaneously. However, a dispatching rule often favors one performance measure only at the expense of other performance measures. In addition, the manufacturing environment usually changes over time. Therefore, the specific dispatching rule employed in such a dynamic environment should be free to change as well.

One of the most notoriously difficult systems for the scheduling community is the job shop system. The strategy of a job shop is based on producing a wide variety of products in very low volumes. Specific customer orders are commonly produced in the job shop manufacturing system. Producing such variable products requires different sequences. In a traditional job shop layout, machines are functionally grouped together. For the case of an actual shop floor, uncertainties (i.e., machine breakdowns, material or tool shortages, transportation delays, etc.) complicate the scheduling problem making it more difficult to solve. Therefore, several assumptions are usually made to simplify the problem (i.e., resources are always available, all the jobs are known in advance, all the operation processing times are known and constant, transportation times are ignored, etc.). However, application of too many such assumptions may result in the treatment of scheduling problems that would be considered unrealistic. That is why the job shop scheduling problems have attracted so much attention over many decades.

1.2. Agent-based approach

Due to the structural rigidity of classical centralized control architectures in manufacturing, the decentralized (or heterarchical) control structure has drawn more attention (Crowe and Stahlman, 1995; Dilts et al., 1991; Duffie and Prabhu, 1994). One of the most important properties of the heterarchical structure is that the decision-making responsibilities are fully distributed to each component of the system. Each component is autonomous and possesses local knowledge that is sufficient to accomplish its own task. The task that a single component is unable to finish alone may require the cooperation of a cluster of components. Communication is a means of establishing such cooperation between the autonomous components. Under the guidance of such a control architecture, the requirements of the next generation of manufacturing systems, such as good fault-tolerance, ease of reconfigurability and adaptability, and agility, can be achieved (Shen and Norrie, 1999).

In recent years, a new paradigm called agent technology has been widely recognized as a promising paradigm for developing software applications able to support complex tasks. From the perspective of a software application, an agent can be viewed as a computational module that is able to act autonomously to achieve its goal (Weiss, 1999; Brenner et al., 1998). A general definition of the software agent is "An agent is a computer system module, capable of acting autonomously in its environment in order to meet its design objectives" (Shen et al., 2000). Wooldridge and Jennings defined an intelligent agent as a hardware or software-based computer system with the properties such as autonomy, social ability, reactivity and pro-activeness (Murch and Johnson, 1998). The idea of agent-based approaches has also offered a promising solution for

controlling future manufacturing systems requiring flexibility, reliability, adaptability, and reconfigurability. Agent technology fits naturally into the decentralized control structure for manufacturing systems because the autonomous component can easily be represented by an agent that is defined as an autonomous, pro-active element with the capability to communicate with other agents (Weiss, 1999). In fact, agents can be used to represent physical shop-floor components such as parts, machines, tools, and even human beings. Under the application of multi-agent systems, each agent is in charge of information collection, data storage, and decision-making for the corresponding shop floor component. A popular scheme to achieve cooperation among autonomous agents is through the negotiation-based contract-net protocol (Smith, 1980). The contract-net protocol provides the advantage of real-time information exchange, making it suitable for shop floor scheduling and control.

1.3. Reinforcement learning

One significant issue for improving an autonomous agent's capability is that of how to enhance the agent's intelligence. Learning is one mechanism that could provide the ability for an agent to increase its intelligence while in operation. Developed in the early 1990s, reinforcement learning (RL) has generated a lot of interest from the research community. As opposed to the popular approach of supervised learning whereby an agent learns from examples provided by a knowledgeable external supervisor (Weiss, 1999), RL requires that the agent learn by directly interacting with the system (its environment) and responding to the receipt of rewards or penalties based on the impact each action has on the system. Although there have been several RL applications demonstrating the usefulness of RL (Sutton and Barto, 1999; Mahadevan and Kaelbing, 1996), its application to manufacturing systems has not been fully explored. The objective of this study was to investigate the application potential of RL on agent-based production scheduling problems. Specifically this paper investigates the application of Q-learning, a widely used RL algorithm to a dispatching rule selection problem on a single machine to determine if it can be used to enable a single machine agent to learn commonly accepted dispatching rules for three example cases in which the best dispatching rules have been previously defined. In addition, experience obtained from this research will be the basis for further investigations into applying RL techniques to more complex agent-based scheduling problems such as dynamic job shop scheduling.

The paper is organized as follows. The next section provides a background of RL and its applications. Section 3 will present the single machine dispatching rule selection problem. The methodologies of this research will be described in Section 4. Section 5 consists of the simulation results and discussion. The conclusions are presented in Section 6.

2. Background

2.1. Introduction of reinforcement learning

In the RL framework, a learning agent must be able to perceive information from its environment. The perceived information is used 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. Based on the new state, there is an immediate reinforcement that 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. Sutton and Barto (1999) defined four key elements for dealing with the RL problems: a policy, a reward function, a value function and a model of the environment. A policy defines the agent's behavior in a given state. A reward function specifies the overall goal of the agent that guides the agent toward learning to achieve the goal. A value function specifies the value of a state or a state-action pair indicating how good it (the state or the state-action pair) is in the long run. A model of environment predicts the next state given the current state and a proposed action.

Besides the above four elements, there is a key assumption in the RL framework. That is, each decision the agent makes is based on the current state that summarizes everything important about the complete sequence of past states leading to it. Some of the information about the complete sequence may be lost, but all that really matters for the future is contained within the current state signal. This is called the Markov property. Therefore, if an environment has the Markov property, then its next state can be predicted given the current state and action. This significant assumption enables the current state to be a good basis for predicting the next state. Under this assumption, the interaction of an agent and its environment can be called a Markov decision process.

For a small RL problem, the estimates of value functions can be represented as a table with one entry for each state or for each state-action pair. However, for a large problem with a large number of states or actions, updating information accurately in such a large table may be a problem. Function approximation is currently a popular method to resolve this issue. Function approximation is an approach generalizing experience from a small subset of examples to develop an approximation over a larger subset. Currently, employing

neural networks is the most popular approach for function approximation in a large RL problem (Sutton and Barto, 1999).

2.2. RL applications to manufacturing systems

Mahadevan et al. (1997a, b) and Das et al. (1999) developed a new model-free average-reward algorithm called SMART for continuous-time semi-Markov decision processes. They applied the SMART algorithm to the problem of optimal preventative maintenance in a production inventory system. In their system, there was a single machine capable of producing multiple types of products with multiple buffers for storing each of the different products. Whenever a job is finished, the machine needs to decide to either undergo maintenance or start another job. Machine maintenance costs and time are less than repair costs and time. In other words, frequent maintenance may be not economical but machine failures resulting from rare maintenance will require more repair costs and time. In their maintenance problem, the state of the system is a 10-dimensional vector of integers that consists of the numbers of five different products manufactured since the last repair or maintenance and the buffer levels of the five products. They compared the maintenance policy learned from SMART to two well-known maintenance heuristics. They found that SMART is more flexible than the two heuristics in finding proper maintenance schedules as the costs are varied. Mahadevan and Theocharous (1998) applied SMART to the problem of optimizing a 3-machine transfer line producing a single product type. The system goal is to maximize the throughput of the transfer line while minimizing WIP and failures. They compared the policy from SMART to the kanban heuristic. Their results showed that the policy learned by SMART requires fewer items in inventory and results in fewer failures than with the Kanban heuristic. Paternina-Arboleda and Das (2001) extended the work of Mahadevan and Theocharous (1998) to deal with a 4-machine serial line and compared SMART to more existing control policies. They examined the system with constant demand rate and Poisson demand rate. Under these two circumstances, SMART outperformed those heuristic policies on average WIP level and average WIP costs.

Zhang and Dietterich (1995) applied RL to job shop type of scheduling problem involving the scheduling of the various tasks that must be performed to install and test the payloads placed in the cargo bay of the NASA space shuttle for each mission. The objective of this problem was to schedule a set of tasks without violating any resource constraints while minimizing the total duration. The scheduling approach Zhang and Dietterich employed was an iterative repair-based scheduling method that started with generating a critical path schedule by ignoring the resource constraints and incrementally repairing the schedule to find a shortest conflict-free schedule. In their system, each state is a complete schedule and each action is a schedule modification. They applied the temporal difference algorithm TD(λ) (an RL algorithm) to this scheduling problem. After taking an action to repair the schedule the scheduler receives a negative reward if the new state still contains constraint violations. This reward function essentially forces the scheduler to not only find a conflict-free schedule but to do it in fewer iterations. The performance of the iterative repair-based procedure with a simulated annealing (SA) method was compared with the one using the TD method. Their results showed that one iteration of the method with TD is equivalent to about 1.8 iterations of the method with SA.

Aydin and Oztemel (2000) proposed an intelligent agent-based scheduling system in which agents are trained by a new RL algorithm they refer to as Q-III. They employed Q-III to train the resource agents to dynamically select dispatching rules. Their state determination criterion consists of the buffer size of the machine and the mean slack time of the queue. The rewards were generated based on some rules from the literature such as SPT is the best rule when the system is overloaded. The thresholds in the rules for determining the systems status were obtained through trial-and-error procedures. Three dispatching rules: SPT, COVERT, and CR, are available for each resource agent to select for their use. The authors compared the proposed scheduling system trained by their RL mechanism to the above three dispatching rules. Their results showed the RL-scheduling system outperformed the use of each of the three rules individually in mean tardiness on most of the testing cases.

2.3. Other applications of RL

More and more work on practical implementations of RL techniques to different fields has been reported. One of the successful stories about RL applications was Tesauro's TD-Gammon (Tesauro, 1995), which was used to play the backgammon game. TD-Gammon was developed based on the TD(λ) algorithm and a multi-layer neural network for function approximation. The latest version of the TD-Gammon was able to play the backgammon game close to the level of the best human player in the world. Another famous application was the elevator-dispatching problem. Modern elevator dispatchers are usually designed heuristically. Crites and Barto (1996) applied the Q-learning to a four-elevator, ten-floor system. Each elevator made its own decision independently to the other elevators. There were some constraints placed on the decisions. The system they dealt with had more than 10^{22} states. Like TD-Gammon, Crites and Barto also employed a neural network to represent the action-value function. Their

RL-based dispatchers outperformed other existing dispatching heuristics on the customer's average waiting time and average squared waiting time. RL also has been widely applied to robotics motion control. Singh and Bertsekas (1997) used the TD(0) algorithm to find dynamic channel allocation policies in cellular telephone systems. Their study showed that RL with a linear function approximation is able to find better dynamic channel allocation policies than two other existing policies. Sutton (1996) applied a RL algorithm, called the Sarsa algorithm, to controlling the motions of a two-link robot. Mahadevan et al. (1997a, b) successfully applied RL to navigating a delivery robot around an indoor office environment.

3. Single machine dispatching rule selection

In this study, we applied the *Q*-learning algorithm to a single-machine scheduling problem. The single-machine production system contains a single buffer for storing jobs awaiting processing. Each job consists of only one operation requiring a variant processing time and the machine can process only one job at a time. If the machine is idle when a job arrives then the job will start processing immediately, otherwise the job will be sent to the buffer. Selection of the next job from the buffer for processing is conducted based on a predefined dispatching rule.

Dispatching rules have been applied to scheduling problems for several decades because of their ease of implementation and low computational requirement. Some dispatching rules are very powerful for finding a good schedule with regard to a specific system objective. Criteria for dispatching rule selection usually take into account the shop status and the overall system objective. In this research, we examined the effect of applying the *Q*-learning technique to the dispatching rule selection problem. Three cases with different system objectives were studied:

a. minimize maximum lateness,
b. minimize number of tardy jobs,
c. minimize mean lateness.

In each case, we have three potential dispatching rules (EDD, SPT, and FIFO) available for use, except for case B, where Hodgson's algorithm is used instead of FIFO. Our goal was to determine if a machine agent is able to learn in each case the best dispatching rules as previously reported by Morton and Pentico (1993).

4. *Q*-learning

The original *Q*-learning algorithm was proposed by Watkins in 1989. 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 pair of state and action (denoted by s and a, respectively). The Q values learned with this algorithm have been proven to converge to the optimal state-action values, Q^* (Watkins and Dayan, 1992). The optimal state-action values for a system represent the optimal policy that the agent intends to learn. The standard procedure of the *Q*-learning algorithm is given as follows:

1. Initialize the $Q(s,a)$ value functions arbitrarily
2. Perceive the current state, s_0
3. Following a certain policy (e.g. ε–greedy), select an appropriate action (a) for the given state (s_0)
4. Execute the selected action (a), receive immediate reward (r), and perceive the next state s_1
5. Update the value function as follows:

$$Q(s_0,a) = Q(s_0,a) + \alpha[r + \gamma \max_b Q(s_1,b) - Q(s_0,a)]. \quad (1)$$

6. Let $s_0 = s_1$
7. Go to step 3 until state s_0 represents a terminal state
8. Repeat steps 2 to 7 for a number of episodes.

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

4.1. State determination criterion

The decisions made by a machine agent are based on the current state of the system. The system state is the basis for organizing the appropriate actions from which an agent will select. Several choices are available for defining the system state; these include such measures as the number of the jobs in the buffer, the number of the tardy jobs in the buffer, or the tardiness or lateness of those jobs. The number of jobs in the buffer (N) and the estimation of the total lateness (ETL) of these jobs were adopted as the state determination criteria. Job lateness was chosen over tardiness since it is able to

distinguish between early jobs (unlike the tardiness measure).

In order to investigate the effects of Q-learning for this dispatching rule selection problem, the Q-learning algorithm was employed only when there were at least two jobs in the buffer. If there is only one job in the buffer, no dispatching rule is needed to determine what job to process next. In this study, 10 distinct states were used to define the framework for a potential policy on dispatching rule selection. These states are differentiated primarily based on ETL. Preliminary experiments indicated that approximately balancing the number of visits to each state may be important (further investigation of this issue is the subject of future tests). Therefore, all the values that were used to define the ranges of the states were adjusted through trial-and-error experiments. Table 1 presents the policy table for use in this study.

When running the Q-learning algorithm, if the previous system state corresponds to a dummy state, updating $Q(s,a)$ is unnecessary because the decision of taking next action is made without considering Q-learning algorithm. However, if the previous state is not a dummy state but the new state is one of the two dummy states (i.e., one job in queue), then an update in this situation must be treated differently because the $Q(s,a)$ values for both dummy states is fixed at zero. The agent should still get the reward/penalty for such decisions, so under these circumstances, the $Q(s,a)$ value is updated using the following equation instead of Eq. (1).

$$Q(s_1,a) = Q(s_1,a) + \alpha r. \tag{2}$$

4.2. Exploration and exploitation

Exploration and exploitation is an important issue when applying the RL algorithm. Exploration means that the agent must try something that has not been done before to get more reward. On the other hands, exploitation is that the agent prefers the actions that were taken before and rewarded. Exploitation may take advantage of guaranteeing a good expected reward in one play, but exploration may provide more opportunities to find the maximum total reward in the long run. One popular approach to deal with this trade-off issue is called ε–greedy method. The ε–greedy method involves selecting, with probability $(1-\varepsilon)$, the action with the best value, otherwise, with small probability ε, an action is selected randomly.

4.3. Reward function

A reward function defines the goal of the learning agent and determines the value of the immediate action based on the perceived state of the environment. Since the learning agent tries to maximize the total reward, the reward function is then essentially used for guiding the learning agent to achieve its goal. In Case A, the system objective is to minimize the maximum lateness. When a job is finished, the lateness of this job (L) is compared to the maximum lateness (ML) of previously completed jobs. If the lateness of the job just completed is greater than the maximum lateness value, the learning machine agent receives a reward of -1. Otherwise, the learning agent receives a reward of $+1$. In other words, selecting a job resulting in a new maximum lateness will result in a penalty of the previous action on the state-action pair table. In Case B, if the currently completed job is tardy, the learning agent receives a penalty of -1, otherwise a reward of $+1$ is assigned. In Case C, the reward function was developed based on the principle that the later a job is finished, the larger its negative reward; likewise the earlier a job is completed (negative lateness) the larger its reward. The details of these three reward functions are shown in Table 2.

Table 1
An example of a 10-state policy table

State	State determination criteria	EDD	SPT	FIFO
Dummy state	$N=0$	0	0	0
Dummy state	$N=1$	0	0	0
1	$N>1$ and ETL <-8	$Q(1,1)$	$Q(1,2)$	$Q(1,3)$
2	$N>1$ and $-8\leqslant$ETL<-5.2	$Q(2,1)$	$Q(2,2)$	$Q(2,3)$
3	$N>1$ and $-5.2\leqslant$ETL<-3	$Q(3.1)$	$Q(3,2)$	$Q(3,3)$
4	$N>1$ and $-3\leqslant$ETL<-0.6	$Q(4,1)$	$Q(4,2)$	$Q(4,3)$
5	$N>1$ and $-0.6\leqslant$ETL<2.5	$Q(5,1)$	$Q(5,2)$	$Q(5,3)$
6	$N>1$ and $2.5\leqslant$ETL<7.7	$Q(6,1)$	$Q(6,2)$	$Q(6,3)$
7	$N>1$ and $7.7\leqslant$ETL<18	$Q(7,1)$	$Q(7,2)$	$Q(7,3)$
8	$N>1$ and $18\leqslant$ETL<45	$Q(8,1)$	$Q(8,2)$	$Q(8,3)$
9	$N>1$ and $45\leqslant$ETL<120	$Q(9,1)$	$Q(9,2)$	$Q(9,3)$
10	$N>1$ and $120\leqslant$ETL	$Q(10,1)$	$Q(10,2)$	$Q(10,3)$

5. Simulation results

A single-machine scheduling simulation was carried out to measure the effects of applying the Q-learning algorithm to train a learning agent to select the proper dispatching rules. The time between job arrivals to the system follows an exponential distribution with a mean of 10. The estimated processing times (EPT) of jobs were uniformly distributed between 7.5 and 8.5. Use of these values result in a mean system utilization is 80% for this system. The due date of the job was set by the following equation:

$$\text{Due date} = \text{Arrival time} + \text{Allowance factor} \times \text{EPT}. \tag{3}$$

The allowance factor in this study was uniformly distributed between 1.5 and 2.5. This due date tightness setting results in the percentage of late jobs being approximately 30%. The real processing time (RPT) of each job was generated using a normal distribution with a mean of EPT and standard deviation of EPT/10. Given the possibility that a normal distribution may generate an extreme value, the RPT values were constrained to be within ± 3 times the standard deviation.

When starting the Q-learning algorithm, the values of the state-action pairs, $Q(s, a)$ can be initialized arbitrarily or assigned specific relative values to represent the confidence in favoring each possible alternative. In this study, all the values of the state-action pairs are initialized to zero since all the actions for each state are assumed to be an equally valid choice. This approach starts the system from a neutral state assuming no a priori knowledge of which dispatching rule is best to use in any situation. Therefore, the system would be required to learn from scratch. Other possible

appogwFxsewT.nUFly nthe

results essentially demonstrate that the machine agent was able to learn to favor the application of the best dispatching rule (as identified for the static single-machine problem, in which all the jobs and their processing times are known in advance, by Morton and Pentico, 1993) for each of the different system objectives.

Table 4 provides the results if each of the rules were used independently as the only rule implemented in the same simulation for each of the three objectives. For instance, it can be observed that for case B, the Hodgson algorithm provided the best performance, followed by SPT and then EDD. The results in Table 4 closely parallel the results in Table 3 demonstrating the performance of the learning mechanism. The EDD heuristic should be the favored for case A, but Table 4 shows that after 500,000 jobs, the performance of both EDD and FIFO are approximately the same. This supports the low value for the EDD rule in Table 3. Overall, we may conclude that the selection percentages of the rules reveal somewhat the relative strengths of each rule for the different performance measures. The rate at which the selection percentages of the rules change during learning process for case A, B, and C is illustrated in Figs. 1–3, respectively.

Table 3
Selection percentages of the dispatching rules (500,000 jobs)

Available dispatching rules	Case A	Case B	Case C
EDD	45.2%	4.8%	7.4%
SPT	16.2%	9.4%	84.2%
FIFO/Hodgson algorithm	38.6%	85.8%	8.4%
Best dispatching rule	EDD	Hodgson	SPT

6. Conclusions

RL has become a very active research field in machine learning. However, applications of RL to manufacturing systems have not been thoroughly explored yet. This study examined the effects of applying the *Q*-learning algorithm to the dispatching rule selection problem for a single machine. Our results confirmed that a machine agent with the *Q*-learning algorithm is able to learn the best rules for different system objectives. The experience from this study provides a good basis for future studies into applying RL techniques to more complicated agent-based scheduling problems like dynamic job shop scheduling.

Table 4
Results of employing the individual dispatching rules (500,000 jobs)

	Case A Max lateness	Case B Num of tardy jobs	Case C Mean lateness
EDD	157.666	247959	3.878
SPT	989.657	165504	3.295
FIFO (cases A and C) Hodgson algorithm (case B)	157.643	128455	3.922

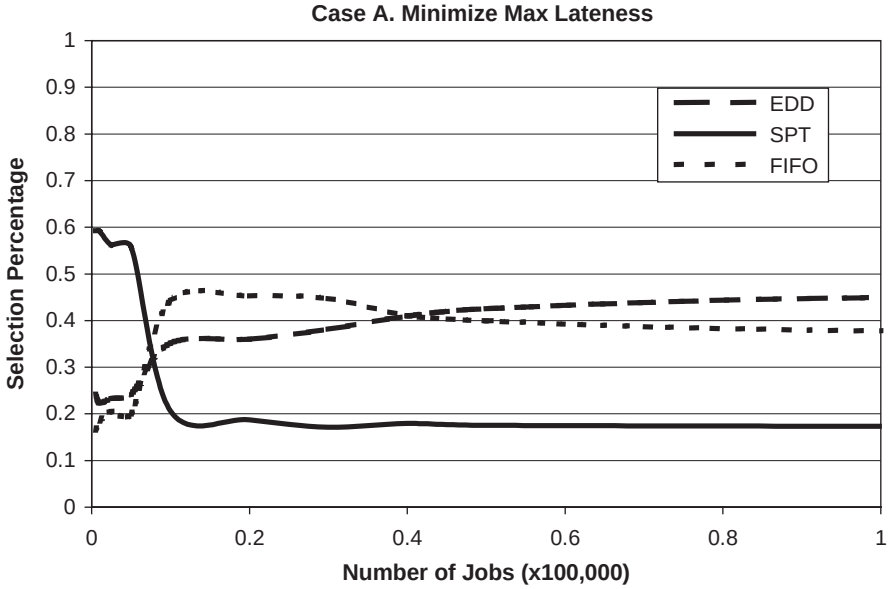

Fig. 1. Rule selection percentages for Case A.

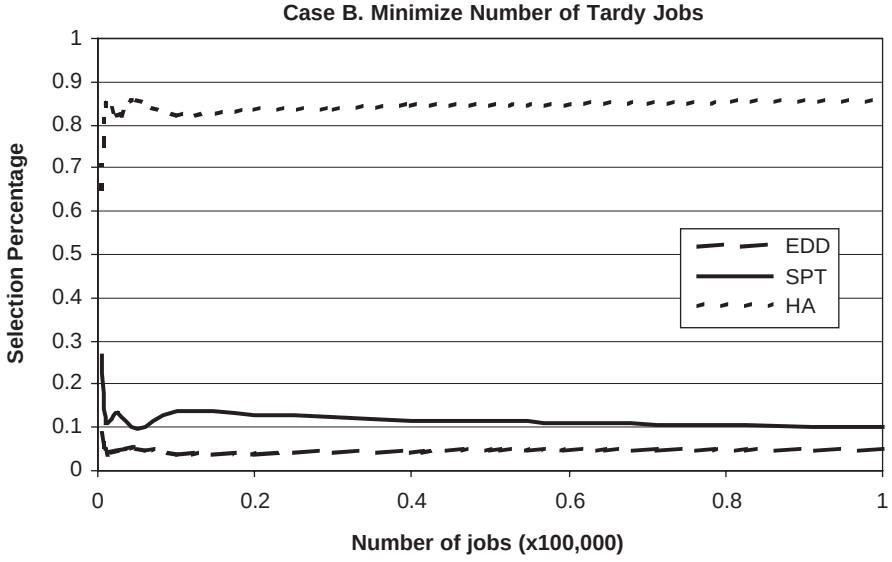

Fig. 2. Rule selection percentages for Case B.

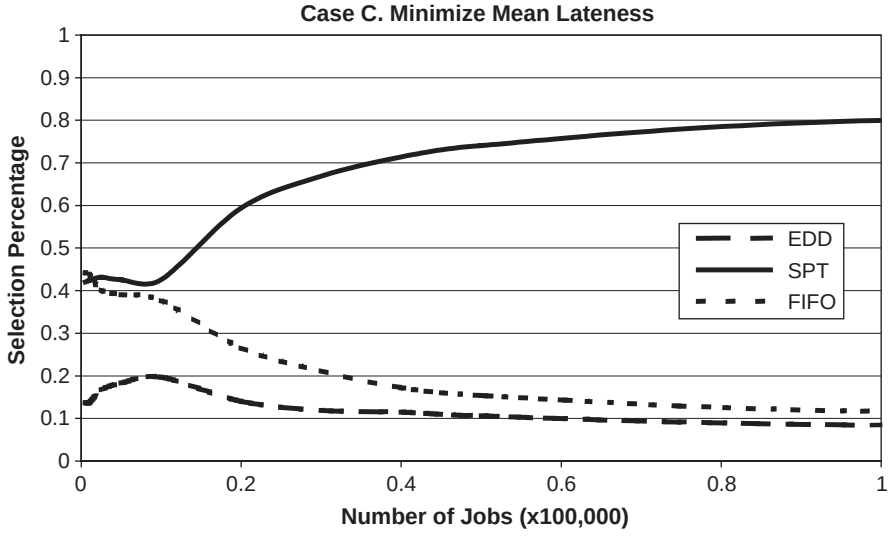

Fig. 3. Rule selection percentages for Case C.

References

Aydin, M.E., Oztemel, E., 2000. Dynamic job-shop scheduling using reinforcement learning agents. Robotics and Autonomous Systems 33, 169–178.

Brenner, W., Zarnekow, R., Witting, H., 1998. Intelligent Software Agents. Springer, Berlin, pp. 19–117.

Brucker, P., 2001. Scheduling Algorithms. Springer, Berlin.

Crites, R.H., Barto, A.G., 1996. Improving elevator performance using reinforcement learning. In: Touretzky, D.S., Mozer, M.C., Hasselmo, M.E. (Eds.), Advances in Neural Information Processing Systems: Proceedings of the 1995 Conference. MIT Press, Cambridge, MA, pp. 1017–1023.

Crowe, T.J., Stahlman, E.J., 1995. A proposed structure for distributed shop floor control. Integrated Manufacturing Systems 6 (6), 31–36.

Das, T.K., Gosavi, A., Mahadevan, S., Marchalleck, N., 1999. Solving semi-Markov decision problems using average reward reinforcement learning. Management Science 45 (4), 560–574.

Dilts, D.M., Boyd, N.P., Whorms, H.H., 1991. The evolution of control architectures for automated manufacturing systems. Journal of Manufacturing Systems 10 (1), 79–93.

Duffie, N.A., Prabhu, V.V., 1994. Real-time distributed scheduling of heterarchical manufacturing systems. Journal of Manufacturing Systems 13 (2), 94–107.

Mahadevan, S., Kaelbling, L.P., 1996. The NSF Workshop on Reinforcement Learning: Summary and Observations. AI Magazine Winter, 89–97.

Mahadevan, S., Theocharous, G., 1998. Optimizing production manufacturing using reinforcement learning. The Eleventh International FLAIRS Conference, AAAI Press, pp. 372–377.

Mahadevan, S., Khaleeli, N., Marchalleck, N., 1997a. Designing agent controllers using discrete-event Markov models. AAAI Fall Symposium on Model-Directed Autonomous Systems, MIT, Cambridge.

Mahadevan, S., Marchalleck, N., Das, T.K., Gosavi, A., 1997b. Self-improving factory simulation using continuous-time average-reward reinforcement learning. Proceedings of the Fourth International Machine Learning Conference, pp. 202–210.

Morton, T.E., Pentico, D.W., 1993. Heuristic Scheduling Systems. Wiley, New York.

Murch, R., Johnson, T., 1998. Intelligent Software Agents. Prentice-Hall, Englewood Cliffs, NJ.

Panwalkar, S.S., Iskander, W., 1977. A survey of scheduling rules. Operations Research 25 (1), 45–62.

Paternina-Arboleda, C.D., Das, T.K., 2001. Intelligent dynamic control policies for serial production lines. IIE Transactions 33, 65–77.

Pinedo, M., 1995. Scheduling Theory, Algorithms, and Systems. Prentice-Hall, Englewood Cliffs, NJ.

Shen, W., Norrie, D.H., 1999. Agent-based systems for intelligent manufacturing: a state-of-the-art survey. Knowledge and Information Systems: an International Journal 1 (2), 129–156.

Shen, W., Norrie, D.H., Barthès, J.-P.A., 2000. Multi-Agent Systems for Concurrent Intelligent Design and Manufacturing. Taylor & Francis, London.

Singh, S.P., Bertsekas, D., 1997. Reinforcement learning for dynamic channel allocation in cellular telephone systems. Advances in Neural Information Processing Systems: Proceedings of the 1996 Conference. MIT Press, Cambridge, MA, pp. 974–980.

Smith, R., 1980. The contract net protocol: high level communication and control in distributed problem solver. IEEE Transactions on Computers 29 (12), 1104–1113.

Sutton, R.S., 1996. Generalization in reinforcement learning: successful examples using spare coarse coding. In: Touretzky, D.S., Mozer, M.C., Hasselmo, M.E. (Eds.), Advances in Neural Information Processing Systems: Proceedings of the 1995 Conference. MIT Press, Cambridge, MA, pp. 1038–1044.

Sutton, R.S., Barto, A.G., 1999. Reinforcement Learning: An Introduction. The MIT Press, Cambridge, MA.

Tesauro, G., 1995. Temporal difference learning and TD-Gammon. Communications of the ACM 38 (3), 58–67.

Watkins, C.J.C.H., Dayan, P., 1992. *Q*-learning. Machine Learning 8, 279–292.

Weiss, G., 1999. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT Press, Cambridge, MA.

Zhang, W., Dietterich, T.G., 1995. A reinforcement learning approach to job-shop scheduling. Proceedings of the 14th International Joint Conference on Artificial Intelligence, pp. 1114–1120.