Proceedings of the 2017 Winter Simulation Conference
W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, eds.

REAL-TIME JOB SHOP SCHEDULING BASED ON SIMULATION AND MARKOV DECISION PROCESSES

Tao Zhang
Shufang Xie
Oliver Rose

Universität der Bundeswehr München
Werner-Heisenberg-Weg 39
Neubiberg, 85577, GERMANY

ABSTRACT

Real-time job shop scheduling is a sequential decision making problem. The main task is to decide which job in a queue should be processed next. The problem can be modeled as a Markov decision process. Jobs in the queue form an action set. Selecting one job to process is regarded as taking an action from the set. A dummy action, which means no job will be selected and the machine will keep idle, is also contained in the set. This removes the no-delay restriction from the problem. The reward function comprises the critical ratio of the selected job and the global job holding cost. Two algorithms, simulation-based value iteration and simulation-based Q-learning, are introduced to solve the scheduling problem from the perspective of a Markov decision process. The simulation explores the state space and accomplishes state transitions. The value function is parameterized and estimated by using a feedforward neural network.

1 INTRODUCTION

In job shops, when a machine becomes idle and several jobs are waiting in front of it, we will decide which job should be processed next. This is a real-time job shop scheduling problem. The decisions are made in real time based on up-to-date information regarding the state of the system. When the level of disturbances in job shops is always high, the real-time scheduling becomes very important. Due to the constraints on the response time, one cannot expect for an optimal or near-optimal decision (Sabuncuoglu and Bayız 2000). Obviously, during the manufacturing process we have to make a huge number of such decisions. These decisions make up a sequence of decisions. Generally, dispatching rules, such as First In First Out (FIFO), Shortest Processing Time (SPT) and Earliest Due Date (EDD), are a simple way to make such decisions (Blackstone, Phillips, and Hogg 1982). However, these dispatching rules are very rough methods and cannot adapt to the changing situation in the manufacturing line (Zhang and Rose 2013). Most times, the sequential decision problem can be described as a Markov decision process (MDP) (Littman 1996). Some applications are already reported (Patrick 2012). For example, Yih and Thesen (1991) formulated the scheduling problem as semi-Markov decision problems and used a non-intrusive 'knowledge acquisition' method to reduce the size of the state space; Gabel and Riedmiller (2007) modeled the job shop scheduling problem by means of a multi-agent reinforcement learning and attached to each resource an adaptive agent that makes its job dispatching decisions independently of the other agents and improves its dispatching behavior by trial and error employing a reinforcement learning algorithm.

However, the applications of Markov decision process in real-time job shop scheduling problems still lack theoretical support. Even for a single agent Markov decision process, lots of issues are still unsolved, for example, how to describe the state of job shops and when to observe the state. The basic principle to

determine the state variables is that the state must be memoryless. For the observation issue, the states at observation points must make up an embedded Markov chain. To possess these properties, some extra constraints may need to be added to the scheduling problems. For example, sometimes the processing times must be subject to exponential distributions. Gabel and Riedmiller (2007) gave some suggestions of state feature selection, but did not consider whether these features are memoryless. The embedded Markov chain is also not mentioned in their work. It is hard to determine whether the job shop scheduling problem was correctly converted into a Markov decision process. Thus, in our study we will simplify the problem and consider only one agent in the Markov decision process. The decision epoch and the state will be explicitly defined. We will also prove that the Markovian property is held in the Markov decision process built from the job shop scheduling problem. Because the state space is very huge, a simulation model is introduced to explore the state space and accomplish the state transitions.

The paper is structured as follows: Markov decision processes are introduced in detail in Section 2. Section 3 shows how we model the scheduling problem as a Markov decision process. Two simulation-based algorithms are proposed in Section 4. An experiment and its results are reported in Section 5. The paper is concluded in the last section.

2 MARKOV DECISION PROCESS

The Markov decision process has two components: a decision maker and its environment. The decision maker observes the state of the environment at some discrete points in time (decision epochs) and meanwhile makes decisions, i.e., takes an action based on the state. The decisions made by the decision maker are then executed in the environment which will find itself in a new state later. As a response to the decision maker, the environment also returns a reward to the decision maker. The goal of the decision maker is to find an optimal way to make decisions so as to maximize the long-term cumulative rewards.

Therefore, the Markov decision process can be described by a five-tuple $MDP = < T, S, A(s), P(s' \mid s,a), R(s' \mid s,a) >$, where T is a set of decision epochs at which the decision maker makes decisions; S is a set of all possible states of the environment; $A(s)$ is a set of possible actions (alternatives) while the state is $s, s \in S$; $P(s' \mid s,a)$ is a set of the probabilities that the state of the environment changes from state s to state s' after action a is taken, which satisfies $\sum_{s' \in S} p(s' \mid s,a) = 1$. $R(s' \mid s,a)$ is a set of rewards that the decision maker obtains after taking action a and the state of the environment changes from state s to state s'. At each decision epoch $t, t \in T$, the decision maker selects an action $a, a \in A(s)$ according to the environment's state $s, s \in S$. As a consequence of its action a, the decision maker receives a numerical reward $r_t, r_t \in \Re$ from the environment, and the state of the environment changes to $s', s' \in S$. A very common goal is that the decision maker tries to find a policy to make decisions so that the sum of the discounted rewards it receives in the future is maximized. The expected discounted reward is (Sutton and Barto 1998),

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

where r is a parameter, $0 \le \gamma \le 1$, called the discount rate.

MDPs can be solved by dynamic programming. Suppose we know $P(s' \mid s,a)$ and $R(s' \mid s,a)$ $\forall s \in S, a \in A(s), s' \in S$ and we wish to calculate an optimal policy π that maximizes the expected discounted reward, where $\pi(s) \in A(s)$ is the action to be taken in state s suggested by policy π. If we

define $Q(s,a)$ as the value of action *a* taken in state s and let $Q(s,a) = \mathrm{E}(R_t \mid S_t = s)$, the following two equations can be derived,

$$Q(s,a) \leftarrow \sum_{s'} p(s' \mid s,a)[r(s' \mid s,a) + \gamma \max_{a'} Q(s',a')]$$

$$\pi(s) = \arg\max_a Q(s,a),$$

To solve the first Bellman equation, we can calculate the optimal value function. The optimal policy can be obtained through the second equation which means that the action with the greatest value $Q(s,a)$ will be taken.

3 MDP VIEW OF REAL-TIME JOB SHOP SCHEDULING

Real-time job shop scheduling is known as a semi-Markov decision process. As we mentioned before, the main concern of the real-time job shop scheduling is to decide which job in the queue should be processed first. This type of decisions has to be made again and again by operators during the manufacturing process. Decisions made now have both immediate and long term effects, and determine the later decision epoch. Obviously, the operators are the decision makers and the job shop is the environment. Selecting one job in a queue to process next corresponds to taking an action in an action set. The goal of the real-time job shop scheduling is not to make good decision at every decision epoch, but to ensure that all decisions together result in a good performance of the job shop.

In order to make the process Markovian, we assume job interarrival times, processing times, and travel times at a machine *m* are all exponentially distributed with mean $1/\lambda_m$, $\mu_{m,p}$, and $\mu_{m,t}$. And the state of system is observed only 1) when a job is released, or 2) a job arrives at an machine, or 3) a machine becomes idle. The five tuple of MDP of job shop scheduling is given as follows.

3.1 Decision epoch and non-decision epoch

Decisions are made while 1) a job arrives at an idle machine; 2) a machine with a non-empty queue becomes idle. We call these points of time decision epochs. Contrarily, when a job is released or arrives at an occupied machine or a machine with an empty queue becomes idle, no decisions are made. We call these moments non-decision epochs. The time between two successive epochs is random.

3.2 State space

The most important art in applying Markov processes is to choose state variables such that the Markovian property holds. Based on queueing network theory, a state of job shops is defined by a set $s = \{s_m\}, \forall m \in M, s_m = (x_m, y_m, z_m)$, where *M* is a set of machines; x_m, y_m, z_m respectively denote number of jobs being travelling to, being waiting in front of, and being processed on machine *m*. Thus, state space $S = \{s\}$. As the exponential assumption about the times and observation at special events, the process $\{s_1, s_2, s_3, \ldots\}$ can be proved to be a Markov process as follows.

Assuming current epoch is at the observation point shown in Figure 1 and current state is *s*. After an action *a* is taken, state becomes to *s'*. Obviously, *s'* is only dependent on *s*, *a*, and the next event. If we can prove that the next occurrence of an event is memoryless, *s'* is only dependent on *s'* and *a.* Because we assume that all times are exponentially distributed, events that just occur at the observation point will occur independently in a memoryless way. For events that do not occur at observation points, we can prove that the remaining time to the next occurrence also has exactly the same exponential distribution as the original (Nelson 2012). So all possible events that may occur next in time have exponentially distributed recurrence times. The next occurred event will be the event with the smallest (remaining) time

to next occurrence, i.e., $\min(X_1, X_2, ...)$, where X is the (remaining) time to the next occurrence. Because $\min(X_1, X_2, ...)$ is also exponential distributed, the memoryless property of the next occurrence is proved.

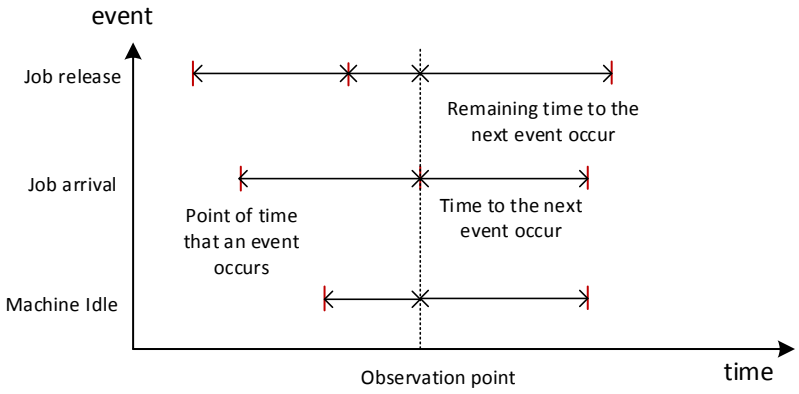

Figure 1: Demonstration of the memoryless property of the next event.

3.3 Action set

Jobs in the related queue at a decision epoch make up an action set. Because our scheduling problem allows machines to be idle while their queues are not empty, a dummy action is also added to the action set. The dummy action means no job is selected to process and the machine keeps idle. At a non-decision epoch, either the related machine is occupied or its queue is empty, so no action is taken. To be consistent with decision epochs, the dummy action is considered to be taken at the non-decision epoch.

3.4 Reward function

The reward, $r(s'|s,a) = (1-\kappa)r_a + \kappa\Delta r_{s'}$, includes two parts, where $0 < \kappa < 1$ indicates the relative importance of the parts. The first part is the reward for the action selection. The second part is the reward for the state sojourning in a time period Δ, where Δ is the time between two successive decision epochs. In our study, r_a adopts the critical ratio of the selected job, $r_a = \varsigma t_{rest} / (\tau_0 + \varsigma t_{raw} - \tau)$, where τ_0 is the release time; τ is current time; t_{raw} denotes the raw processing time; t_{rest} denotes the remaining raw processing time; ς is a real number, $\varsigma > 1$. For the dummy action, $r_a = 0$. The second part is considered to be the cost of holding jobs in the system, $r_{s'} = -\upsilon \sum_{m \in M} (x_m + y_m + z_m)$,where υ is the price of holding one job per time unit.

3.5 Transition function

First, let us consider one machine *m* from the network and analyze it for itself. Jobs entering the machine area are either from release sources or from its predecessors. The rates at which jobs enter the machine area from each source are $\lambda_m^1, \lambda_m^2, ..., \lambda_m^{N_{m,in}}$, where $N_{m,in}$ is the number of the sources and predecessors. Assuming that current state of the machine is (x_m, y_m, z_m), after an action is taken the state changes to a new state with a certain probability. At any epoch, three types of events may occur. Sometimes only one event occurs; sometimes two or more events may occur together. Different combinations of the events are related to different states. There are 7 possible states after an action is taken. The corresponding probabilities of the states will be the probabilities that the related events occur first. The probability that an event occurs before other events can be calculated through the rates

$P(X_k = \min(X_a, X_b, ...)) = \lambda_k / (\lambda_a + \lambda_b + ...)$. For example, the probability that a job arrive event occurs first is $y_m \mu_{m,t} / \nu(x_m, y_m, z_m)$, where $\nu(x_m, y_m, z_m) = \lambda_m + y_m \mu_{m,t} + z_m \mu_{m,p}$, $\lambda_m = \lambda_m^1 + \lambda_m^2 + ... + \lambda_m^{N_{m,in}}$. The possible new states and their probabilities are listed in Table 1 in which $\lambda_m^\Gamma = \sum_{I \in A_m} \prod_{i \in I} \lambda_m^i$, where A_m is a set of sets which contain any *n* numbers from 1 to $N_{m,in}$.

Table 1: New states and their probabilities after an action is taken in state (x, y, z).

Action	Next Events	Next state	Probability
Dummy action	*n* job released, $0 < n \le N_{m,in}$	$(x+n,y,z)$	$\lambda_m^\Gamma / \nu^n(x,y,z)$
	Job arrival	$(x-1,y+1,z)$	$y\mu_t / \nu(x,y,z)$
	Machine idle	$(x,y,z-1)$	$z\mu_p / \nu(x,y,z)$
	Job release & Job arrival	$(x+n-1,y+1,z)$	$\lambda_m^\Gamma y\mu_t / \nu^{n+1}(x,y,z)$
	Job release & Machine idle	$(x+n,y,z-1)$	$\lambda_m^\Gamma z\mu_p / \nu^{n+1}(x,y,z)$
	Job arrival & Machine idle	$(x-1,y+1,z-1)$	$y\mu_t z\mu_p / \nu^2(x,y,z)$
	Job release & Machine idle& Job arrival	$(x+n-1,y+1,z-1)$	$\lambda_m^\Gamma y\mu_t z\mu_p / \nu^{n+2}(x,y,z)$
Non-dummy action (z=0,y>0)	*n* job released, $0 < n \le N_{in}$	$(x+n,y-1,z+1)$	$\lambda_m^\Gamma / \nu^n(x,y-1,z+1)$
	Job arrival	$(x-1,y,z+1)$	$(z+1)\mu_p / \nu(x,y-1,z+1)$
	Machine idle	$(x,y-1,z)$	$(y-1)\mu_t / \nu(x,y-1,z+1)$
	Job release & Job arrival	$(x+n-1,y,z+1)$	$\lambda_m^\Gamma (y-1)\mu_t / \nu^{n+1}(x,y-1,z+1)$
	Job release & Machine idle	$(x+n,y-1,z)$	$\lambda_m^\Gamma (z+1)\mu_p / \nu^{n+1}(x,y-1,z+1)$
	Job arrival & Machine idle	$(x-1,y,z)$	$(y-1)\mu_t (z+1)\mu_p / \nu^2(x,y-1,z+1)$
	Job release & Machine idle& Job arrival	$(x+n-1,y,z)$	$\lambda_m^\Gamma (y-1)\mu_t (z+1)\mu_p / \nu^{n+2}(x,y-1,z+1)$

Now, we extend our focus to the whole network of machines. Since all machines are considered, at each epoch the possible events will be the observed events that may occur at all machines. To calculate the probability that one event occurs prior to all other events, we have to rewrite function $\nu(x_1, y_1, z_1, x_2, y_2, z_2, ...) = \lambda_1 + y_1\mu_{1,t} + z_1\mu_{1,p} \quad + \lambda_2 + y_2\mu_{2,t} + z_2\mu_{2,p} + ...$. Thus the probability that, for example, a job will finish on machine *m* at the next epoch will be $z_m \mu_{m,p} / \nu(x_1, y_1, z_1, x_2, y_2, z_2, ...)$. Other probabilities can be computed in a similar way. In addition, some events occuring at two successive machines may be identical, like an event that a machine becomes idle after a job finishes and the job release event at its successor. So in order to simplify the problem, only one of them should be considered. Note, that the number of possible events will become very large if the number of machines increases. Thus, it is impossible to predefine the transition function in this case. Even a problem with only 10 machines is already very hard to be solved.

4 SIMULATION-BASED ALGORITHM

Because the state space is quite large and sometimes the transition function is hard to be obtained, we use simulation to explore the state space and accomplish the state transitions.

4.1 Parameterized value function

The task of the MDP is to calculate optimal values of all state-action pairs. The optimal policy can be derived from these optimal values by means of any greedy algorithm. However, for the MDP with a large state space, only a part of the states is explored and their values are computed. In order to generalize the obtained values to unexplored states, a parameterized value function $Q(s,a)$ is introduced. Consequently, states and actions must be factorized. In our problem, the states are already well factorized. So, just for actions, three factors are adopted: waiting time t_w, progress ratio ϑ, and processing time t_p. For a dummy action, the values of these three factors are all set to -1. Thus $Q(s,a) = Q(x_1, y_1, z_1, ..., x_{|M|}, y_{|M|}, z_{|M|}, t_p, \vartheta, t_w)$. Moreover, feed forward networks can be used for any kind of input to output mapping. A feed forward network with one hidden layer and enough neurons in the hidden layers can fit any finite input-output mapping problem (Ilonen, Kamarainen, and Lampinen 2003). Hence, we use it to map the relationship between $Q(s,a)$ and (s,a). The factors of states and actions are inputs and the value of the state-action pairs are the output of the neural network.

4.2 Simulation-based Value Iteration Algorithm

Value iteration is a very common algorithm to compute $Q(s,a)$ (Puterman 2014). For our problem, we improved it with simulation and neural networks. Through simulation, we can find the most frequent states and their approximate probabilities of occurence after an action is taken. With this information, we will update the value of the state-action pair. The state, action, and updated value form a training pattern which is used to update the neural network. The algorithm is given in Figure 2. Note that once an action is taken at a state, we run sub-simulations many times so as to explore the state space. According to the times that a state appears, its probability is computed. Obviously, if the simulation runs long enough and the sub-simulations run enough times, a near optimal value function can be achieved.

```
Initialize neural network Q arbitrarily
Start simulation
Once an arrival or completion event occurs, Do
    Pause simulation
    Select an action through the neural network, a* = arg max_{a∈A(s)} Q(s,a)
    Carry out the action in the simulation
    Run sub-simulations enough times from current state till the next event, Do
        Build a set S', which contains possible states after the action is taken
        Build sets A(s'), which contain possible actions at state s'
        Calculate the probability p(s'|s,a) for each state in S'
        Calculate the immediate reward r(s'|s,a) for each state in S'
        Q(s,a) ← Σ_{s'∈S'} {p(s'|s,a)[r(s'|s,a) + γ max_{a'∈A(s')} Q(s',a')]}
        Update neural network Q with pattern <s, a, Q(s, a)>
    End Do
    Resume simulation
End simulation when a terminate condition is met
```

Figure 2: Simulation-based value iteration algorithm.

4.3 Simulation-based Q-learning

In the simulation-based value iteration algorithm, in order to calculate the probabilities, the sub-simulations are very time-consuming. Q-learning avoids the probability estimation and explores only one state at a time. Therefore, we also tried Q-learning in our study. Like the previous algorithm, the Q-learning algorithm is also improved with simulation and neural networks. The improvement algorithm is shown in Figure 3.

Initialize neural networks Q arbitrarily

Set previous state $s^- = null$ and previous action $a^- = null$

Start simulation

Once an arrival or completion event occurs, Do

$s \leftarrow$ current state, $A(s) \leftarrow$ current action set

If s^- is not null Then

$$Q(s^-, a^-) \leftarrow Q(s^-, a^-) + \alpha[r(s \mid s^-, a^-) + \gamma \max_{a' \in A(s)} Q(s, a') - Q(s^-, a^-)],$$

where α is learning rate, $0 < \alpha < 1$

Update neural networks Q with pattern $< s^-, a^-, Q(s^-, a^-) >$

End If

Select and take an action through ε -greedy derived from the neural networks,

$$a = \begin{cases} \arg\max_{a^* \in A(s)} Q(s, a^*) & \zeta < \varepsilon \\ random(A(s)) & otherwise \end{cases}$$

$s^- \leftarrow s$, $a^- \leftarrow a$

End simulation when a terminate condition is met

Figure 3: Simulation-based Q-learning.

5 EXPERIMENTS

The approach is applied in a small example manufacturing system, shown in Figure 4. The system contains 5 machines and produces 2 products (Pa and Pb) with 2 operation flows. The release interarrival times of Pa and Pb are exponentially distributed with mean 5 and 8 time units. The travel times and processing times also follow exponential distributions. The objective is to minimize the cycle time.

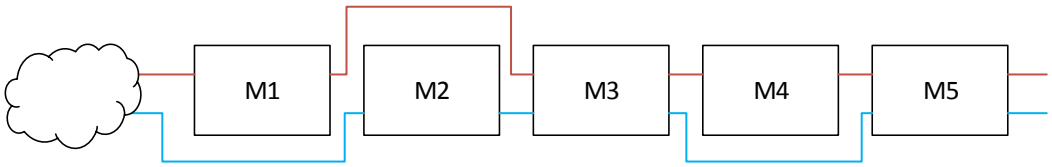

Figure 4: A job shop with two types of product.

A discrete event simulation model is created. The model is used in both learning phase and validation phase. The simulation-based Q-learning algorithm is adopted. The learning rate and discount factor are set to $\alpha = 0.1, \gamma = 0.1$. The parameters in the reward function have the following values: $\kappa = 0.5, \varsigma = 2.5, \upsilon = 0.025$. A neural network is built with 18 inputs, 1 hidden layer, and 5 neurons in the hidden layer. Details of the neural network training are neglected here. In the learning phase, the simulation runs for 10 hours of real time. In the validation phase, the simulation runs for 100000 time

units of simulation time. The trained neural network is simply used as a decision rule which calculates priority values of jobs in the queue. The decisions are made on the basis of the values, i.e., the job with the greatest value will be selected. If a dummy action has the greatest value, no job is selected and the machine will be kept idle. Several common dispatching rules including FIFO, Shortest Processing Time (SPT), Longest Processing Time (LPT), and Critical Ratio (CR) are also tested by the simulation and the results are compared with our results in the MDP column shown in Table 2. Our approach results in the shortest average cycle time and the lowest average WIP level.

Table 2: Comparison between the proposed approach and some dispatching rules.

Items \ Approach		FIFO	SPT	LPT	CR $\varsigma = 2.5$	MDP
Avg. Cycle Time	Pa	107.92	110.69	120.13	93.70	**97.65**
	Pb	107.74	114.73	110.04	124.45	**110.42**
	Summary	107.85	112.25	116.27	105.49	**102.55**
Avg. WIP	Pa	21.51	22.21	24.19	18.58	**19.62**
	Pb	13.37	14.41	13.75	15.44	**13.80**
	Summary	34.89	36.63	37.93	34.02	**33.42**
Number of Finished Jobs	Pa	19929	20051	20113	19828	**20082**
	Pb	12410	12563	12476	12386	**12488**
	Summary	32339	32614	32589	32224	**32570**

6 CONCLUSIONS

The real-time job shop scheduling problem is viewed as a Markov decision process. The decision maker learns the scheduling knowledge gradually through interacting with the job shop environment. We proved that the defined state process $\{s_1, s_2, s_3, \cdots\}$ is a Markov process. Because the state space is quite large, we introduced two simulation-based algorithms to solve the MDP: simulation-based value iteration algorithm and simulation-based Q-learning. In order to improve the generalization ability of our approach, a feed-forward neural network is introduced to map state-action pairs to their values. The experimental results show that our approach performs better than some other decision rules.

REFERENCES

Blackstone, J. H., D. T. Phillips, and G. L. Hogg. 1982. "A State-of-the-Art Survey of Dispatching Rules for Manufacturing Job Shop Operations." *The International Journal of Production Research* 20 (1):27-45.

Gabel, T., and M. Riedmiller. 2007. "Adaptive Reactive Job-Shop Scheduling with Reinforcement Learning Agents." *International Journal of Information Technology and Intelligent Computing* 24 (4):1-30.

Ilonen, J., J.-K. Kamarainen, and J. Lampinen. 2003. "Differential Evolution Training Algorithm for Feed-Forward Neural Networks." *Neural Processing Letters* 17 (1):93-105.

Littman, M. L. 1996. "Algorithms for Sequential Decision Making." Ph.D. Thesis, Department of Computer Science, Brown University. Providence, RI, USA.

Nelson, B. L. 2012. *Stochastic Modeling: Analysis and Simulation*. New York: Dover Publications.

Patrick, J. 2012. "A Markov Decision Model for Determining Optimal Outpatient Scheduling." *Health Care Management Science* 15 (2):91-102.

Puterman, M. L. 2014. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Hoboken, New Jersey: John Wiley & Sons, Inc.

Sabuncuoglu, I., and M. Bayız. 2000. "Analysis of Reactive Scheduling Problems in a Job Shop Environment." *European Journal of Operational Research* 126 (3):567-586.

Sutton, R. S., and A. G. Barto. 1998. *Reinforcement Learning: An Introduction*. Cambridge, London: MIT Press.

Yih, Y., and A. Thesen. 1991. "Semi-Markov Decision Models for Real-Time Scheduling." *International Journal of Production Research* 29 (11):2331-2346.

Zhang, T., and O. Rose. 2013. "Intelligent Dispatching in Dynamic Stochastic Job Shops." In *Proceedings of the 2013 Winter Simulation Conference*, edited by S.-H. Kim R. Pasupathy, A. Tolk, R. Hill, and M. E. Kuhl, 2622-2632. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

AUTHOR BIOGRAPHIES

TAO ZHANG is a Research Assistant and Ph.D. student working on production planning and scheduling at the Department of Computer Science of the Universität der Bundeswehr München, Germany. From 2007 to 2009 he received his Master in Metallurgical Engineering with the subject of production planning and scheduling in iron and steel industry from Chongqing University, China. He is involved in modeling and simulation of complex system and intelligent optimization algorithms. His email address is tao.zhang@unibw.de.

SHUFANG XIE is a Research Assistant and PhD student at Universität der Bundeswehr at the Chair of Modeling and Simulation. Her focus is on simulation-based scheduling and optimization of production systems. She has received her M.S. degree in Metallurgical Engineering from Chongqing University, China. Her email address is shufang.xie@unibw.de.

OLIVER ROSE holds the Chair for Modeling and Simulation at the Department of Computer Science of the Universität der Bundeswehr, Germany. He received a M.S. degree in applied mathematics and a Ph.D. degree in computer science from Würzburg University, Germany. His research focuses on the operational modeling, analysis and material flow control of complex manufacturing facilities, in particular, semiconductor factories. He is a member of INFORMS Simulation Society, ASIM, and GI. His email address is oliver.rose@unibw.de.