Proceedings of the 2018 Winter Simulation Conference
M. Rabe, A.A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, eds.

REAL-TIME BATCHING IN JOB SHOPS BASED ON SIMULATION AND REINFORCEMENT LEARNING

Tao Zhang
Shufang Xie
Oliver Rose

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

ABSTRACT

Real-time batching in job shops decides 1) whether to start processing a batch or to wait for more jobs joining the batch, and 2) which batch should be processed first. It is addressed as a sequential decision-making problem and formalized based on Markov decision processes. By adding a dummy batch, which means no batches are selected and all batches wait for additional jobs, the first decision-making is generalized to the second. A simulation-based neural fitted Q learning is introduced to solve the Markov decision processes and build a decision maker. The well-trained decision maker decides which batch in a batch list should be processed first at each decision epoch. The experiment results show that the proposed approach outperforms some other decision rules.

1 INTRODUCTION

In job shops, batching occurs when a machine can process several jobs simultaneously. Usually, the jobs belonging to the same family can form a batch. They are started and completed together. Because of the machine capacity, the maximal batch size is limited. In the real world, jobs arrive at the machine randomly. Always forming full batches may lower machine utilization due to the long waiting times. It may also delay some other urgent jobs. On the other hand, small batches will reduce the actual machining capacity. The real-time batching tries to find a good timing to start each batch so as to balance the utilization and capacity. More specifically, the real-time batching answers the question: should we start to process a batch or wait for more jobs joining the batch? In the multiple job family environment, several batches may form in front of the machine. We should answer another sequencing question: which batch should be processed first. In the study, both questions will be considered.

Substantial research work has gone into the scheduling with batching area and made significant progress. Brucker et al. (1998) addressed a problem of scheduling *n* jobs on a batching machine to minimize regular scheduling criteria that are non-decreasing in the job completion times. A dynamic programming algorithm solves the problem. Ahmadi et al. (1992) propose a heuristic and establish an upper bound on the worst-case performance ratio of the heuristic for the problem. Potts and Kovalyov (2000) even review the literature on scheduling with batching, giving details of the basic algorithms and referencing other significant results. All of them consider the scheduling with batching as a deterministic problem and use optimization algorithms to solve it. However, in practice, many things in job shops are stochastic, such as job arrivals, processing times, interruptions, and so on. Their algorithms will not be suitable for such a stochastic environment. Considering the complexity of the real job shops, making a schedule in advance becomes very hard. Therefore, practitioners prefer to consider the scheduling with batching as decision-making problems, i.e., the real-time batching. They developed lots of decision rules to answer the two questions. For the first question, the most essential decision rule is the minimum batch size (MBS), which is introduced by Neuts (1967). The batching machine starts to process a batch only when a minimum number of jobs are present in the batch. Another common rule is the maximum waiting time (MWT) under which the machine starts to process a non-full batch whenever the batching time reaches the maximum. Considering the future job arrivals, Glassey and Weng (1991) present a dynamic batching heuristic (DBH) which dynamically calculates the number of the arrivals that each batch should wait for. Fowler et al. (1992) modify the DBH rule to consider only the next arrival and decide whether to start the batch now or to wait for the next arrival. The biggest disadvantage of this rule is that we must determine some parameters (such as MBS and MWT) or forecast the future job arrivals ahead. For the second question, decision rules, such as First In First Out (FIFO), Shortest Processing Time (SPT), and Earliest Due Date (EDD), are very common answers (Blackstone et al. 1982). However, these decision rules are very rough methods and cannot adapt to the changing situation in the job shops (Zhang and Rose 2013). Gabel and Riedmiller (2007) adopt an alternative view by modeling the job-shop sequencing problems as multi-agent reinforcement learning problems. In fact, they interpret job-shop sequencing problems as Markov decision processes and attach 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 application of Markov decision processes in real-time job shop sequencing problems still lacks 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. Zhang et al. (2017) simplify the problem and consider only one agent in the Markov decision process. The decision epoch and the state are explicitly defined. The study proves that the Markovian property is held in the Markov decision process built from the job shop sequencing problems. The problem is solved by simulation and reinforcement learning.

In this study, we will extend the study further and try to answer both the sequencing and batching questions together. The paper is structured as follows: Markov decision processes and reinforcement learning are introduced in Section 2. Section 3 shows how we model the real-time batching problem as a Markov decision process. A simulation-based neural fitted Q learning algorithm is proposed in Section 4. An experiment and its results are reported in Section 5. The paper is concluded in the last section.

2 REINFORCEMENT LEARNING AND MARKOV DECISION PROCESSES

In this section, we will give a brief introduction to reinforcement learning and Markov decision processes. Most contents, like concepts and notations, are from the book by Sutton and Barto (1998). For more information, please refer to this book.

2.1 Reinforcement Learning

Reinforcement learning is learning how to act in different situations through interacting with the environment so as to maximize a numerical reward signal. The learner is not told which actions to take but instead must discover which actions yield the most reward by trying them or by the experience of the past trials. Actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. Trial-and-error search and delayed reward are the two most important distinguishing features of reinforcement learning.

2.2 Markov Decision Processes

Markov Decision Processes (MDPs) are a mathematically idealized form of the reinforcement learning problem for which precise theoretical statements can be made. A Markov decision process has two components: a decision maker and its environment. The decision maker observes the state of the environment at 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 $MDP = <T, S, A(s), P(s' | s,a), R(s' | s,a)>$, 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 $s, s \in S$; $P(s' | s,a)$ 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

$$\sum_{s' \in S} p(s' | s,a) = 1 .$$

$R(s' | s,a)$ 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 $P(s' | s,a)$ and $R(s' | s,a)$ $\forall s \in S, a \in A(s), s' \in S$, 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 $Q(s,a) = \mathrm{E}(R_t | S_t = s)$, the following two equations can be derived:

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

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

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 BATCHING

Real-time batching is a semi-Markov decision process. As we mentioned before, the main concern of the real-time batching is to decide whether to start processing a batch and which batch in the queue should be processed first. These types of decisions have 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. Apparently, the operators are the decision makers, and the job shop is the environment. Making a decision corresponds to taking an action. The goal of the real-time batching is that all decisions together result in the good long-term performance of the job shop. The five-tuple of MDP of the real-time batching is given as follows.

3.1 Decision Epoch and Non-decision Epoch

The state of the job shop is periodically observed in a fixed interval. This can prevent some batches from waiting too long. The state is also observed 1) when a job is released, 2) a job arrives at a machine, or 3) a machine becomes idle. This can make sure that the decision maker responds to state changes quickly. Decisions are made while 1) a job arrives at an idle batching machine; 2) a batching machine with a non-empty queue becomes idle; or 3) at a periodical observation point where a batching machine is idle, and the queue is not empty. We call these points of time decision epochs. Contrarily, no decisions are made 1) when a job is released or arrives at an occupied machine, 2) a machine with an empty queue becomes idle, or 3) at a periodical observation point where a batching machine is not idle, or the queue is empty. We call these moments non-decision epochs. The time between two successive epochs is random.

3.2 Action Set

Batches in the related queue at a decision epoch plus a dummy batch make up a batch list. Selections of a batch from the list form the action set. Selecting the dummy batch means no batches will be processed and the batches wait for more jobs. The dummy batch merges two decision-making problems to one selection problem. Each action is quantified by features of the concerned batch. For now, the following features are considered: processing time of the batch, waiting time of the batch, and quotient of current batch size and maximal batch size. For the dummy batch, the values of these features are all -1. At a non-decision epoch, either the related machine is occupied, or its queue is empty, so no action is taken. The dummy action is considered to be taken at the non-decision epoch to be consistent with decision epochs.

3.3 State Space

Based on queueing network theory, a state of job shops is defined by a set $s=\{s_m\}$, $\forall m \in M$, $s_m=(x_m, y_m, z_m, t_m)$, where M is a set of machines; x_m, y_m, z_m respectively denote the number of jobs being traveling to, being waiting in front of, and being processed on machine m. t_m is the estimated remaining time that machine m becomes idle. If the machine is already idle, $t_m=0$. In the multiple job family environment, the traveling jobs and the waiting jobs are grouped according to their families. x and y are two sets of the number of jobs in each group, $x_m=\{x_m^f\}$, $y_m=\{y_m^f\}$, where f donates a job family, $f \in F$. F is a set of all job families. Finally, state space $S=\{s\}$. Obviously, the state space is infinite.

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 selected action. The second part is the reward for the state sojourning in a period, where Δ is the time between two successive decision epochs. Both parts are normalized values. In our study, before the normalization,

$$r_a=(n_b/n_{mb})/(t_w/t_p),$$

where t_w is the waiting time of the batch; t_p is the processing time of the batch; n_b is the current batch size; n_{mb} is the maximal batch size. This formula indicates that selecting big batches with short waiting times will get more rewards. For the dummy batch, $r_a=0$. The second part is considered to be the cost of holding jobs in the system, before the normalization,

$$r_{s'}=-\upsilon\sum_{m\in M}(x_m+y_m+z_m),$$

where υ is the price of holding one job per time unit. In the multiple job family environment,

$$r_{s'} = -\upsilon \sum_{m \in M} \left(\sum_{f \in F} x_m^f + \sum_{f \in F} y_m^f + z_m \right).$$

3.5 Transition Function

The job shop is a discrete event system. After an action is taken, the next state depends only on the next event. The probability of the new state equals the probability that the related events occur first. For example, in a two machine and one job family system, the current state is $s = (x_{m1}, y_{m1}, z_{m1}, t_{m1}, x_{m2}, y_{m2}, 0, 0)$. After a batch is selected to process on machine *m2* and the next event is the batch completion at machine *m1*, the new state $s' = (x_{m1}, y_{m1}, 0, 0, x_{m2}, y_{m2} - n_b, n_b, t_p)$. The probability that the state changes from s to s' equals the probability that the batch completion event at machine *m1* occurs first. Even though the number of event types is very small in the job shop, the number of possible new states will be huge considering the places where the events occur and the concurrent events. Therefore, there is no way to list all probabilities that one state changes to another after an action is taken.

4 SIMULATION-BASED NEURAL FITTED Q LEARNING

As we mentioned before, once optimal values of all state-action pairs are computed from the Bellman equation, the optimal policy can be derived from these optimal values by means of any greedy algorithm. However, the state transition probabilities must be given in the equation, which is impossible in the study. Therefore, we introduce Q learning (Watkins and Dayan 1992) to solve the MDP problem.

4.1 Neural Fitted Q Learning

Q learning is a model-free reinforcement learning technique, which means we do not need to know details of the model, like the transition probabilities. Q learning can be used to find an optimal action-selection policy for any given MDPs. It works by learning an action-value function that ultimately gives the expected reward of taking a given action in a given state and following the optimal policy thereafter. The Q learning is defined by

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[r(s' \mid s,a) + \gamma \max\nolimits_{a' \in A(s')} Q(s',a')].$$

To use the Q learning, we face another problem. The MDP of the real-time batching has an infinite state space. But, only a part of the states can be explored, and their values can be computed. In order to generalize the obtained values to unexplored states, a parameterized value function $Q(s,a)$ is needed. Because a feed-forward network with one hidden layer and enough neurons in the hidden layers can fit any finite input-output mapping problem (Ilonen et al. 2003), we use it to map the relationship between $Q(s,a)$ and (s,a). The states and actions are inputs, and the value of the state-action pairs are the output of the neural network. This algorithm is called neural fitted Q learning (shown in Figure 1) which was first proposed by Riedmiller (2005).

4.2 Simulation-based Neural Fitted Q Learning (SNFQ)

The neural fitted Q learning always tries to observe and interact with the real system. In our study, it is not practicable to interact with the job shops. Thus, we adopt a simulation model of the job shops instead. The algorithm will observe and interact with the simulation. The simulation explores the state space and accomplishes the state transitions. The improved algorithm is shown in Figure 2. First, the neural network Q is initialized arbitrarily. Then we start the simulation. Once a job arrival or completion event or a cyclic observation event occurs, we pause the simulation and collect data including the old and current states, the action taken before, the rewards, and the action set in the current state. The value of the action $Q(s^-, a^-)$ is re-calculated by

```
Initialize neural networks Q arbitrarily, let $i$=0
While $i$ < Max iteration number, do
    Initialize current state $s$
    While $s \neq s^f$, do
        Take action $a$ using a policy derived from Q
        Observe new state $s'$, $r(s'|s,a)$, and action set $A(s')$ in state $s'$
        Generate a training pattern
            Input=<s, a>
            Output=$r(s'|s,a)+\gamma \max_{a' \in A(s')} Q(s',a')$
        Train neural networks Q using the pattern
        $s \leftarrow s'$
    End
    $i$=$i$+1
End
Output a deterministic policy, $\pi$, so that
        $\pi(s) = \arg\max_a Q(s,a)$
```

Figure 1: Neural fitted Q learning.

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

where $r(s\,|\,s^-,a^-)$ is obtained from the reward function and $Q(s,a')$ is computed from the neural network. Meanwhile, the neural networks are updated by the new pattern, i.e., $< s^-,\ a^-, Q\left(s^-,\ a^-\right) >$. After that, we make decisions according to the updated neural networks and resume the simulation. The algorithm terminates when the simulation ends.

We can see that the initial value of action obtained from the initial neural networks is improved after the update by means of the new information from the simulation. The initial neural networks are also improved after the training process by using the improved date, i.e., the new value of action. In the following round, the value of action will be further improved by not only the new information from the simulation but also the improved neural networks. Naturally, the longer the simulation run, the more states are explored, and the better the neural network is trained.

5 EXPERIMENTS

The proposed approach is applied in a small example job shop, shown in Figure 3. The job shop contains three machines (M1, M2, and M3) and produces two products (Pa and Pb) with two process flows. All machines are batching machines. The maximal batch sizes are 3, 3, and 8 jobs respectively. Machine M3 can process both Pa and Pb, and only the jobs of the same product can be put in a batch. The interarrival release times of Pa and Pb are exponentially distributed with mean 5 and 8 time units. The travel times and processing times follow normal distributions. The objective is to minimize the average cycle time.

We create a discrete event simulation model of the example shop. The model is used in both learning phase and validation phase. Since a decision we made now has insignificant influence on the future in our case, the discount factor γ is set to a small value of 0.1. The parameters in the reward function have the following values: $\kappa = 0.8, \upsilon = 0.025$, because if we can always keep the WIP level as low as possible, the objective (the cycle time) will be minimized. Thus, we gave κ a big value of 0.8, which means the reward is mainly dependent on the holding cost of jobs in the system. A neural network is built with 18 inputs which is similar to the study of Zhang and Rose (2013), and one hidden layer, because one hidden layer is sufficient for the large majority of problems. According to an empirically-derived rule that the optimal size of the hidden layer is usually between the size of the input and size of the output layers, we put six neurons in the hidden layer. In the learning phase, the simulation runs for 5 hours of real time. In the validation phase, the simulation runs 5 times for 100,000 units of simulation time each.

```
Initialize neural networks Q arbitrarily
Set previous state $s^- = null$ and previous action $a^- = null$
Start simulation
Once an arrival or completion or cyclic observation event occurs, Do
    $s \leftarrow$ current state, $A(s) \leftarrow$ current action set (batch list)
    If $s^-$ is not null, Then
        $Q(s^-, a^-) \leftarrow r(s \mid s^-, a^-) + \gamma \max_{a' \in A(s)} Q(s, a')$,
        Update neural networks Q with pattern $< s^-, a^-, Q(s^-, a^-) >$,
    End If
    Select a batch through $\varepsilon$-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 2: Simulation-based neural fitted Q learning.

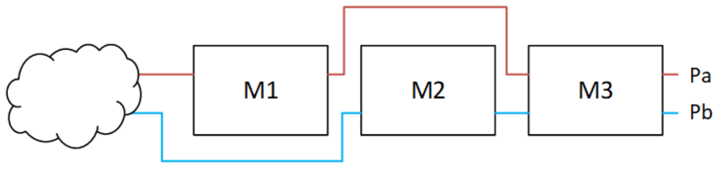

Figure 3: A job shop with three batching machines and two types of product.

The trained neural network is simply used as a decision rule, which calculates priority values of batches (including the dummy batch) in the queue. The decisions are made on the basis of the values, i.e., the batch with the greatest value will be selected. If a dummy batch has the highest value, no batch is selected, and the machine will be kept idle and wait for more jobs. The proposed algorithm (SNFQ) is compared with the MBS, MWT, and DBH rules. All of the sequencing decisions are made by the FIFO rule. The result is shown in Table 1. Our approach SNFQ produces the shortest average cycle time and the lowest average WIP level. The insignificant variation in the number of finished jobs is mainly caused by stochastic processes of the job release and is ignored in the comparison.

6 CONCLUSIONS

The real-time batching is a sequential decision-making problem, which can be addressed as a five-tuple Markov decision process. The study defines the five-tuple of MDP of the real-time batching, including the action set, state space, reward function, transition function, and decision epochs. Because of the infinite state space, a neural fitted Q learning is introduced to solve the MDP. The neural fitted Q learning can generalize the visited state to an unknown state. A simulation model is built to replace the real job shop. The algorithm interacts with the simulation and learns the batching knowledge gradually. The experimental results show that our approach performs better than conventional decision rules. However, several questions remain unanswered at present: 1) are there any other reward functions better than the current one; 2) how can we obtain the optimal parameters of the approach, such as the interval of the cyclic observations in the simulation, the discount factor in Q learning, the number of neurons in the neural networks, and so on; 3) how do simulation, Q learning, and neural networks affect each other and the whole performance of the approach? A further study with more focus on these questions is, therefore, suggested.

Table 1: Comparison between the proposed approach and selected decision rules.

Items \ Approach		MBS+FIFO	MWT+FIFO	DBH+FIFO	**SNFQ**
Avg. Cycle Time	Pa	40.32	51.98	39.32	**38.87**
	Pb	68.66	73.91	68.47	**66.09**
	Summary	51.13	60.62	50.65	**49.40**
Avg. WIP	Pa	8.06	10.40	7.86	**7.77**
	Pb	8.58	9.24	8.56	**8.26**
	Summary	16.65	19.63	16.42	**16.04**
Number of Finished Jobs	Pa	20068	19967	19937	**20009**
	Pb	12477	12547	12536	**12503**
	Summary	32645	32514	32473	**32512**

REFERENCES

Ahmadi, J. H., R. H. Ahmadi, S. Dasu, and C. S. Tang. 1992. "Batching and Scheduling Jobs on Batch and Discrete Processors." *Operations Research* 40(4):750-763.

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.

Brucker, P., A. Gladky, H. Hoogeveen, M. Y. Kovalyov, C. N. Potts, T. Tautenhahn, and S. L. Van De Velde. 1998. "Scheduling a Batching Machine." *Journal of scheduling* 1(1):31-54.

Fowler, J. W., G. L. Hogg, and D. T. Phillips. 1992. "Control of Multiproduct Bulk Service Diffusion/Oxidation Processes." *IIE transactions* 24(4):84-96.

Gabel, T., and M. Riedmiller. 2007. "Adaptive Reactive Job-Shop Scheduling with Reinforcement Learning Agents." *International Journal of Information Technology and Intelligent Computing* 2(4), w/o page numbers.

Glassey, C. R., and W. W. Weng. 1991. "Dynamic Batching Heuristic for Simultaneous Processing." *IEEE transactions on semiconductor manufacturing* 4(2):77-82.

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.

Neuts, M. F. 1967. "A General Class of Bulk Queues with Poisson Input." *The Annals of Mathematical Statistics* 38(3):759-770.

Potts, C. N., and M. Y. Kovalyov. 2000. "Scheduling with Batching: A Review." *European Journal of Operational Research* 120(2):228-249.

Riedmiller, M. 2005. "Neural Fitted Q Iteration–First Experiences with a Data Efficient Neural Reinforcement Learning Method." In *Proceedings of the European Conference on Machine Learning*, edited by J. Gama et al., 317-328. Berlin, Heidelberg: Springer.

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

Watkins, C. J., and P. Dayan. 1992. "Q-Learning." *Machine learning* 8(3-4):279-292.

Zhang, T., and O. Rose. 2013. "Intelligent Dispatching in Dynamic Stochastic Job Shops." In *Proceedings of the Winter Simulation Conference*, edited by R. Pasupathy, et al., 2622-2632. Piscataway, New Jersey: IEEE.

Zhang, T., S. Xie, and O. Rose. 2017. "Real-Time Job Shop Scheduling Based on Simulation and Markov Decision Processes." In *Proceedings of the 2017 Winter Simulation Conference (WSC)*, edited by W. K. V. Chan et al., 3899-3907. Piscataway, New Jersey: IEEE.

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 systems and intelligent optimization algorithms. His email address is tao.zhang@unibw.de.

SHUFANG XIE is a Research Assistant and Ph.D. student at the 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 an 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, especially semiconductor factories. He is a member of the INFORMS Simulation Society, ASIM, and GI. His email address is oliver.rose@unibw.de.