Simulation in Produktion
und Logistik 2019
Matthias Putz & Andreas Schlegel (Hrsg.)
Wissenschaftliche Scripten, Auerbach 2019

Online Single Machine Scheduling Based on Simulation and Reinforcement Learning

Online-Single-Machine-Scheduling basierend auf Simulation und Reinforcement Learning

Shufang Xie, Tao Zhang, Oliver Rose, Universität der Bundeswehr München, München (Germany), shufang.xie@unibw.de, tao.zhang@unibw.de, oliver.rose@unibw.de

Abstract: In this paper, an online single machine environment is used to investigate the application potential of reinforcement learning for job shop scheduling problems, to quantify states and actions, to define reward functions, and to examine their influences on the performance of the algorithm through plenty of experiments. The findings from this research are expected to be the basis for further investigations into applying reinforcement learning to more complex job shop environments in the future.

1 Introduction

During manufacturing, when a machine becomes idle and several jobs are queuing, waiting to be processed, we need to decide which job should be chosen according to up-to-date information about the system. This is called online scheduling. The goal is that all decisions combined will result in a well performaning manufacturing system, with shorter cycle times, less tardiness, and so on. Reinforcement learning is a particular method to learn how to make decisions in different situations through interacting with the environment in order to maximise a numerical reward. The decisionmaker is not told which decisions to take but instead must discover which decisions yield the most reward by trying them or by experience from the past trials. Lots of research studies utilise reinforcement learning to learn scheduling knowledge and, afterward, make decisions according to that knowledge. Aydin et al. (2000) use reinforcement learning to train agents so the they can select the most appropriate priority rule according to the shop conditions in real-time. Gabel and Riedmiller (2008) have modeled the online scheduling problem by means of multi-decision maker reinforcement learning and attached an adaptive decision-maker to each resource that makes its job dispatching decisions independently from the other decision-makers and improves its dispatching behavior by trial and error. Zhang et al. (2017) used simulation and reinforcement learning to solve this problem. Jobs in the queue form an action set. Selecting one job to process is regarded as taking

action on the set. 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. The simulation explores the state space and accomplishes state transitions. The value function is parameterised and estimated by using a feedforward neural network.

However, the applications of reinforcement learning in online scheduling still lack theoretical support and lots of issues are still unsolved, for example, how to quantify the states and actions, how to define the reward function and when the neural network is used to fit the value function, how to ensure the convergence of the learning, etc. Moreover, it is also of interest how far current solutions are from the optimum . In this study, some of these questions are answered by carrying out a huge amount of experiments. Since the job-shop environment is too complicated to carry out these analyses, a single-machine environment will be adopted. The advantage of the single-machine environment is that the optimal solution/rules are already known in some cases. Thus, the algorithms can always be compared with the optimal rules and it will be known how much room is left for algorithmic improvements. Since it is difficult to connect to a real system, a simulation model of the single-machine environment is built. The learning procedure interacts with the simulation and learns the scheduling knowledge.

The paper is structured as follows: The single machine scheduling problem and reinforcement learning are introduced in Section 2. In Section 3, the simulation-based neural-fitted Q-learning is applied to solve the problem. The states, actions, and reward functions are defined and quantified in different ways in Section 4. Based on the definitions, experiments are designed and carried out in Section 5. The paper is concluded in the last section.

2 Fundamental Principles

In this chapter, two fundamental principles are explained: Online single machine scheduling and reinforcement learning.

2.1 Online Single Machine Scheduling

Single machine scheduling has already received substantial attention in the last decades. This kind of problem is important because of its own intrinsic value as well as its role as a building block for more generalised and complex problems (Kgupta and Kyparisis, 1987). Moreover, in a multi-machine environment, the schedules may be made around a bottleneck or expensive machine, or sometimes an entire production line may be treated as a single machine for scheduling purposes. This problem has been studied under different constraints with various objectives.

In this study, we will focus on a preemptive non-delay single-machine-scheduling problem with two objectives: minimising total completion time and minimising maximal lateness. The machine cannot be kept idle if jobs are waiting and the machine allows for preemptions of jobs by other jobs that arrive later. To simplify the problem, breakdowns, set up, and some other characteristics frequently appearing in real manufacturing systems are being ignored. Jobs arrive over time and the number of jobs is unlimited.

The Shortest Processing Time first (SPT) rule is the optimal rule for optimising total completion time. According to this rule, whenever a job arrives, we compare the processing time of the job with the remaining processing time of the job currently on the machine. If the new job has a smaller processing time, we preempt the job on the machine and replace it with the job that has just arrived. The Earliest Due Date (EDD) rule is the optimal rule for optimising maximal lateness. In this rule, whenever a job arrives, we compare the due date of the job to the due date of the job that is currently on the machine. If the new job has an earlier due date, we preempt the job on the machine and replace it with the new job. This approach will be compared with the rules later.

2.2 Reinforcement Learning

Reinforcement learning is a general class of machine learning algorithms. The paradigm of reinforcement learning handles learning in sequential decision-making problems. The elements of the reinforcement learning problem can be formalised using the Markov Decision Process (MDP) framework (Sutton and Barto,1998).

The MDP 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. $R(s' \mid s,a)$ 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 a new state $s', s' \in S$. A very common goal for the decision-maker is to find a policy to make decisions so that the sum of the discounted rewards $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ received in the future is maximised, while r is the discount rate and $0 \le \gamma \le 1$. To solve this problem, the value of action/action-value function is defined as $Q(s, a)$ which denote how well is action *a* being selected in state *s*. If the state space and action set are small enough and the transition probabilities are given, dynamic programming can calculate optimal values of actions for all state-action pairs. When making decisions, we always take the action with the highest value. However, the sizes of the online scheduling problems are usually very big, and the probabilities are very difficult to be obtained. It is impossible to iterate over all possible state-action pairs (Zhang et al., 2017). Therefore, Q-learning is introduced to solve the problem in the next section.

3 Simulation-based Neural-fitted Q-learning

Before a detailed explanation of simulation-based neural-fitted Q-learning, the reinforcement learning technique Q-learning and neural-fitted Q-learning will be introduced first.

3.1 Q-learning

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

$$Q(s,a) \leftarrow (1-\alpha)\mathrm{Q(s,a)} + \alpha[\mathrm{r(s'|s,a)} + \gamma max_{deA(s')} Q(s'a')] \quad (1)$$

To use Q-learning, we face another problem. The MDP of online scheduling has an infinite state space. Only a part of the states can be explored. To generalise the obtained values to unexplored states, a parameterised 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 which was first proposed by Riedmiller (2005).

3.2 Simulation-based Neural-fitted Q-learning

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 real job shops. Thus, we adopt a simulation model of 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 Algorithm 1.

First, the neural network Q is initialised arbitrarily. Then we start the simulation. Once a job arrival or completion event occurs, the simulation is paused to 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, where $r(s \mid s^-,a^-)$ is obtained from the reward function and $Q(s,a')$ is computed from the neural network. Meanwhile, the neural network is updated by the new pattern, i.e., $< s^-,\ a^-, Q\left(s^-,\ a^-\right) >$. After that, we make decisions according to the updated neural network 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 network is improved after the update by means of the new information from the simulation. The initial neural network is also improved after the training process by using the improved data, i.e., the new value of an 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 network. Naturally, the longer the simulation runs, the more states are explored, and the better the neural network is trained.

Algorithm 1: *Simulation-based neural-fitted Q-learning*

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, $A(s) \leftarrow$ jobs in the queue
  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 job through ε-greedy,

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

  $s^- \leftarrow s$, $a^- \leftarrow a$
End simulation when a terminate condition is met

4 State, Action, and Reward in Online Single Machine Scheduling

As mentioned in the previous section, the state and action are inputs of the neural network. The value of action calculated from the reward is the output of the neural network. Thus, we must quantify the state and action and define the reward function before we start the simulation-based Q-learning. Since there are lots of ways to define the state and action, we will list the definitions for as many as possible in this section.

4.1 State of Single Machine Environment

The single machine environment includes one machine and one queue. The state of the environment is formed by the states of the machine and queue. Because the machine is always idle when making decisions, the state of the machine can be ignored here. The state of a queue can be some statistic attributes of the jobs in the queue:

- Number of jobs
  - Number of jobs in the queue, i.e., queue length (QL)
  - Number of tardy jobs in the queue (QNTJ)
- Processing time (QPT)
  - Sum of processing times of jobs in the queue, i.e., queue time (SumQPT)
  - Minimal processing time of jobs in the queue (MinQPT)

  - Maximal processing time of jobs in the queue (MaxQPT)
  - Variance of processing times of jobs in the queue (VarQPT)
  - Average processing time of jobs in the queue (AvgQPT)
  - QPTs = {AvgQPT, MinQPT, MaxQPT, VarQPT}
- Due date (QDD)
  - Sum of Due dates of jobs in the queue (SumQDD)
  - Earliest due date of jobs in the queue (MinQDD)
  - Latest due date of jobs in the queue (MaxQDD)
  - Variance of due dates of jobs in the queue (VarQDD)
  - Average due date of jobs in the queue (AvgQDD)
  - QDDs = {AvgQDD, MinQDD, MaxQDD, VarQDD}

We can also calculate some new items from these basic data, for example, waiting times (QWT) and slack times (QST). Both can have the same statistic attributes to others.

4.2 Action

The action is the selection of one job from the queue. The attributes of the job can describe actions, like

- Release time (RT)
- Waiting time (WT), WT=NOW-RT
- Processing time (PT)
- Due date (DD)
- Slack time (ST), ST=DD-NOW-PT.

4.3 Reward/Cost Function

The reward function cannot tell how we select jobs but addresses what we want. So, the reward can be the objective achieved so far or some other KPIs. Some of them are listed below.

- Total completion time so far (TCT)
- Average WIP level so far (WIP)
- Total tardiness so far (TT)
- Total lateness so far (TL)
- Max lateness so far (ML)
- Number of tardy jobs so far (NTJ)

When the simulation starts (NOW is 0), TCT, ML, and NTJ are 0 and WIP is the same as current QL. The reward can also be the attributes of the queue or the job, like QL, QNTJ, the sum of QST (SumQST), PT, DD, and so on.

5 Experiments

A single machine model is built in which jobs arrive randomly and the interarrival times follow an exponential distribution. The processing times and due dates are constant and only known after the jobs arrive. The experiments will be carried out on this model.

5.1 Experimental Design

Separate experiments are designed for the two objectives. For minimizing total completion time, the state can be described by QL, QPT, QWT, or their combination. The action can be distinguished from their PT, WT, or both. The reward functions have -TCT, -ML, -NTJ, and -SumTCT, where SumTCT=-Σ (Big Number-PT). For this objective, the value of an action depends only on the PT. So, two cases (A15 and A16) without state input are added. In total, 16 cases are created, shown in Table 1.

Table 1: *Experiment cases of the objective: minimising total completion time*

Case No.	Description (State|| Action|| Reward)	Case No.	Description (State|| Action|| Reward)
A1	QL||PT||-TCT	A2	QL||PT||-NTJ
A3	QL||PT||-ML	A4	QL||PT||-SumTCT
A5	QL||PT,WT||-TCT	A6	QL||PT,WT||-SumTCT
A7	QL,QPTs||PT||-TCT	A8	QL,QPTs||PT||-SumTCT
A9	QL,QPTs||PT,WT||-TCT	A10	QL,QPTs||PT,WT||-SumTCT
A11	QL,QPTs,QWTs||PT||-TCT	A12	QL,QPTs,QWTs||PT||-SumTCT
A13	QL,QPTs,QWTs||PT,WT||-TCT	A14	QL,QPTs,QWTs||PT,WT||-SumTCT
A15	0 ||PT||-TCT	A16	0 ||PT||-SumTCT

For minimizing maximal lateness, QL, QDD, and QWT are selected for the state, DD, and WT for the action. The reward functions include -ML, -TCT, -NTJ, -TL, and -TT. Like Table 1, three cases (B15, B16, and B17) without state input are included. Finally, 17 cases are shown in Table 2.

Table 2: *Experiment cases of the objective: minimising maximal lateness*

Case No.	Description (State|| Action|| Reward)	Case No.	Description (State|| Action|| Reward)
B1	QL||DD||-ML	B2	QL||DD||- NTJ
B3	QL||DD||-TCT	B4	QL||DD||-TT
B5	QL||DD,WT||-ML	B6	QL||DD,WT|| -TT
B7	QL,QDDs||DD||-ML	B8	QL,QDDs||DD|| -TT
B9	QL,QDDs||DD,WT||-ML	B10	QL,QDDs||DD,WT|| -TT
B11	QL,QDDs,QWTs||DD||-ML	B12	QL,QDDs,QWTs||DD|| -TT
B13	QL,QDDs,QWTs||DD,WT||-ML	B14	QL,QDDs,QWTs||DD,WT|| -TT
B15	0 ||DD||-ML	B16	0 ||DD||-TT
B17	0 ||DD||-TL		

5.2 Experimental Results

In the experimental design, there are 33 cases. Simulation-based Q-learning is run for each case. When the simulation ends, the neural networks are supposed to be well trained. We run the simulation again for one month of simulated time. The decisions in the simulation are made according to the values of action calculated from the trained neural networks. After one month, we extract total completion time or maximal lateness from the simulation results and compare them with results from several rules, including SPT, First-In-First-Out (FIFO), Longest Processing Time first （LPT）, Random (RAND), Critical Ratio (CR), EDD, and Least Slack (LS). The results are listed in Table 3 and Table 4.

Table 3: *Experiment results of the objective: minimising total completion time*

Case No.	Total Completion Time	Case No.	Total Completion Time	Case No.	Total Completion Time
SPT	**10326.66**	LPT	14097.67	EDD	12198.46
FIFO	12286.88	RAND	12173.18	CR	14097.67
LS	12226.50	A1	10982.54	A2	11823.91
A3	13961.45	**A4**	**10785.84**	A5	11091.54
A6	11333.39	A7	13018.59	A8	12983.78
A9	13123.76	A10	12998.43	A11	13564.69
A12	12875.24	A13	12432.37	A14	12084.31
A15	10845.76	**A16**	**10667.34**		

Table 4: *Experiment results of the objective: minimising maximal lateness*

Case No.	Maximal Lateness	Case No.	Maximal Lateness	Case No.	Maximal Lateness
SPT	32.77	LPT	34.01	**EDD**	**0.93**
FIFO	32.77	RAND	33.91	CR	31.60
LS	0.98	B1	31.38	B2	2.52
B3	31.21	**B4**	**1.14**	B5	32.09
B6	3.87	B7	33.64	B8	34.49
B9	33.34	B10	33.88	B11	34.56
B12	33.50	B13	32.30	B14	33.13
B15	32.43	**B16**	**1.76**	B17	32,76

From Table 3, we can see that the optimal rule (i.e., SPT) generates the smallest total completion time while LPT leads to the worst case. Cases A1, A4, A15, and A16 outperform the other designed cases. Looking inside of these four cases, they all use PT to describe the action and one (QL) or no variable represents the state. A15 and

A16 without state information input are even better than A1 and A4. Comparing the four with cases A7-A14 which have 6-11 input variables, we can figure out that the more variables we input, the worse results we get. Since we know the value of action is only up to the PT, considering other irrelevant inputs only worsen the situation. The performances of A15 and A16 prove this point. If we compare cases A1-A4 which have the same definition of the state and action, but different reward functions, it is obvious that A1 and A4 with objective-related reward functions are better than the other two with the reward functions regardless of the objective.

When looking at cases B4, B6, B8, B12, and B14 in Table 4, we can support a similar assertion about the inputs (quantification of the state and action). The reason is that unnecessary inputs can make the state space even larger. Exploring the state space becomes more difficult. In addition, if we check the cases B1, B5, B7, B9, B11, B13, and B15 with reward function -ML, even though the function is highly related to the objective: minimising maximal lateness, it shows very bad performance. When we adopt -TL in case B17, the performance is still bad. Thus, objective-related reward functions are not always the right choice.

6 Conclusion

In this paper, a single machine environment is used to investigate the application potential of reinforcement learning in job shop environments. First, we modelled a single machine environment. Then, we quantified the state and action in many ways and defined many reward functions to investigate their influence on the algorithm. Lots of experimental cases are designed and carried out. From the results, we find that unnecessary inputs can only make the state space even larger and worsen the performance. So, taking only the necessary information from states and actions is always the right step. The results also show that the objective-highly-related reward functions may not perform well, sometimes, even worse than other reward functions. It would be better to try and compare more reward functions and select the best one.

References

Aydin, M. and Öztemel, E.: Dynamic Job-shop scheduling Using Reinforcement Learning Agents. Robotics and Autonomous Systems, 33 (2000) 2-3, pp. 169-178.

Gabel, T.; Riedmiller, M.: Adaptive Reactive Job-Shop Scheduling with Reinforcement Learning Agents., International Journal of Information Technology and Intelligent Computing 24 (2008) 4.

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

Kgupta, S.; Kyparisis, J.: Single Machine Scheduling Research. Omega,15 (1987) 3, pp. 207-227.

Riedmiller, M.: Neural Fitted Q Iteration–First Experiences with a Data Efficient Neural Reinforcement Learning Method. In: Gama, J., Camacho, R., Brazdil, P., Jorge, A., Torgo, L. (Eds.): Proceedings of the 16th European Conference on Machine Learning (ECML) Porto (Portugal), 3.-7. October 2005, pp. 317-328.

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

Zhang, T.; Xie, S.; Rose, O.: Real-time job shop scheduling based on simulation and Markov decision processes. In: Chan, W. K. V; D'Ambrogio, A.; Zacharewicz, G.; Mustafee, N.; Wainer, G.; Page, E. (Eds.): Proceedings of the Winter Simulation Conference, (WSC), Las Vegas (USA), 3.-6. December 2017, pp. 3899-3907.