Proceedings of the 2020 Winter Simulation Conference
K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, eds.

DEEP Q-NETWORK MODEL FOR DYNAMIC JOB SHOP SCHEDULING PPROBLEM BASED ON DISCRETE EVENT SIMULATION

Yakup Turgut
Cafer Erhan Bozdag

Department of Industrial Engineering
Istanbul Technical University
Macka
Istanbul, 34367, TURKEY

2020 Winter Simulation Conference (WSC) | 978-1-7281-9499-8/20/$31.00 ©2020 IEEE | DOI: 10.1109/WSC48552.2020.9383986

ABSTRACT

In the last few decades, dynamic job scheduling problems (DJSPs) has received more attention from researchers and practitioners. However, the potential of reinforcement learning (RL) methods has not been exploited adequately for solving DJSPs. In this work deep Q-network (DQN) model is applied to train an agent to learn how to schedule the jobs dynamically by minimizing the delay time of jobs. The DQN model is trained based on a discrete event simulation experiment. The model is tested by comparing the trained DQN model against two popular dispatching rules, shortest processing time and earliest due date. The obtained results indicate that the DQN model has a better performance than these dispatching rules.

1 INTRODUCTION

The most basic job shop scheduling problem (JSP) can be defined as follows: there are n jobs with a set of operations that need to be processed by m machines. Each job has technical precedence order on m machines. The key problem is sequencing n jobs on m machines by minimizing a measure of performance. Although the JSP seems to be a simple problem in concept, it is a difficult problem to solve optimally because it is an NP-hard combinatorial optimization problem (Morton and Pentico 1993; Garey, Johnson, and Sethi 1976). Priority rules or dispatching rules are widely used for JSP because they are simple to use and have low time complexity. The priority rules work as a function, assign a number to each job, and determine which one to choose to process on a machine from waiting jobs. In the event of a tie, the job with the smallest number is selected (Gere Jr 1966). On the other hand, computational algorithms and heuristics based on dispatching rules have been extensively applied to solve the JSP optimally or near-optimally (Chen and Matis 2013; Branke and Pickardt 2011; Cheng et al. 1996).

The JSP is called static JSP if the information related to the jobs to prepare the schedule is known beforehand. If it is not known, it is called the dynamic JSP. Because real-world scheduling problems are dynamic, researchers and practitioners have begun to focus on solving dynamic JSP rather than static JSP over the past several decades. It can be seen that solving dynamic scheduling problems requires more effort than solving static scheduling problems.

DJSPs can be interpreted as sequential decision processes and job scheduling decisions can be improved by training an agent based on reinforcement learning algorithms. The literature review in Section 2 indicates that there has been little work on employing RL algorithms for dynamic JSP. This study aims to construct an RL model based on DQN (Deep Q-Network) to sequence jobs dynamically In this study deep Q-network (DQN) model is applied to train an agent to learn how to schedule the jobs dynamically by minimizing the delay time of jobs. The DQN model is trained based on a discrete event simulation experiment. The

results of the proposed model show that the trained agent is able to schedule jobs that dynamically arrive at the system while considerably reducing the scheduling cost according to two popular dispatching rules.

The remainder of this paper is organized as follows. Section 2 presents the literature concerning dynamic JSP. Section 3 describes the application of the DQN model to dynamic JSP. The simulation experiments and results are discussed in Section 4. Section 5 concludes the paper and discusses directions for future work.

2 DYNAMIC JOB SHOP SCHEDULIG

Dynamic JSP has been studied over several decades. Ramasesh (1990) proposed a classification scheme of scheduling problems. Based on this scheme, the JSP can be classified as a static model and dynamic model according to the nature of job arrivals at the shop and the nature of requirements. The author defines DJSP as a dynamic model where jobs arrive at the shop randomly and intermittently over time.

DJSP is not restricted to the nature of job arrivals. DJSP can be extended by including other dynamic events such as machine breakdowns, changes in due dates, order cancellation, the arrival of urgent orders (Baykasoğlu and Karaslan 2017; Fang and Xi 1997; Suwa and Sandoh 2007; Larsen and Pranzo 2019). Baykasoğlu and Karaslan (2017) developed a new approach based on a greedy randomized adaptive search procedure to solve DJSP by taking into account the sequence-dependent setup times in addition to the above dynamic events. With the addition of these dynamics events, DJSP is also called a dynamic rescheduling problem in the literature. Larsen and Pranzo (2019) introduced a simulation-based framework for dynamic rescheduling problems.

A major research effort focuses on developing new methods to solve the job shop scheduling problems optimally or near-optimally at a reasonable time. Mohan et al. (2019) presented a survey on the development of dynamic job shop scheduling, they indicated that scheduling methods can be classified as precise methods and approximate methods. Simulation has been extensively used because of its capabilities of handling the complexity of DJSP (Kiran and Smith 1984; Turker et al. 2019; Zhang et al. 2017; Xiong et al. 2017). Ramasesh (1990) presented a literature survey of simulation studies for DJSP. Turker et al. (2019) proposed a decision support system based on simulation to increase the performance of dispatching rules, they indicated that the DSS improves the following performance measures: workstation utilization, number of tardy jobs, amount of waiting time of jobs.

On the other hand, reinforcement learning can be used as an alternative approach for solving DJSPs (Zhang and Dietterich 1995; Gabel and Riedmiller 2008; Aydin and Öztemel 2000; Riedmiller and Riedmiller 1999; Gabel and Riedmiller 2012; Wang and Usher 2005). Wang and Usher (2005) investigated the application of Q-learning algorithm for dispatching rule selection problem on a single machine. The results of the study demonstrate the potential of RL for application to the JSP. Gabel and Riedmiller (2012) tackled DJSP as a sequential decision-making problem and applied policy gradient RL for solving dynamic and static JSP. Aydin and Öztemel (2000) introduced an intelligent agent-based system for dynamic scheduling problems. These researchers developed Q-III learning algorithm to select the most appropriate dispatching rule dynamically. Gabel and Riedmiller (2008) considered the dynamic job shop scheduling problem as a multi-agent learning problem.

3 DQN Model for DYNAMIC JOB SHOP SCHEDULING

In this section, we give the basics of the DQN model and describe how it is adopted to solve dynamic JSP.

3.1 DQN Model

DQN (Deep Q-Network) was developed by Mnih et al. (2015) to handle the reinforcement learning problems with large state space. A successful Atari game implementation of the algorithm have inspired the RL community to use it in other domains. It is a function approximation method and a deep convolutional neural network is used as a value function approximation. Value function parameterized as $Q(s,a;\theta_i)$.

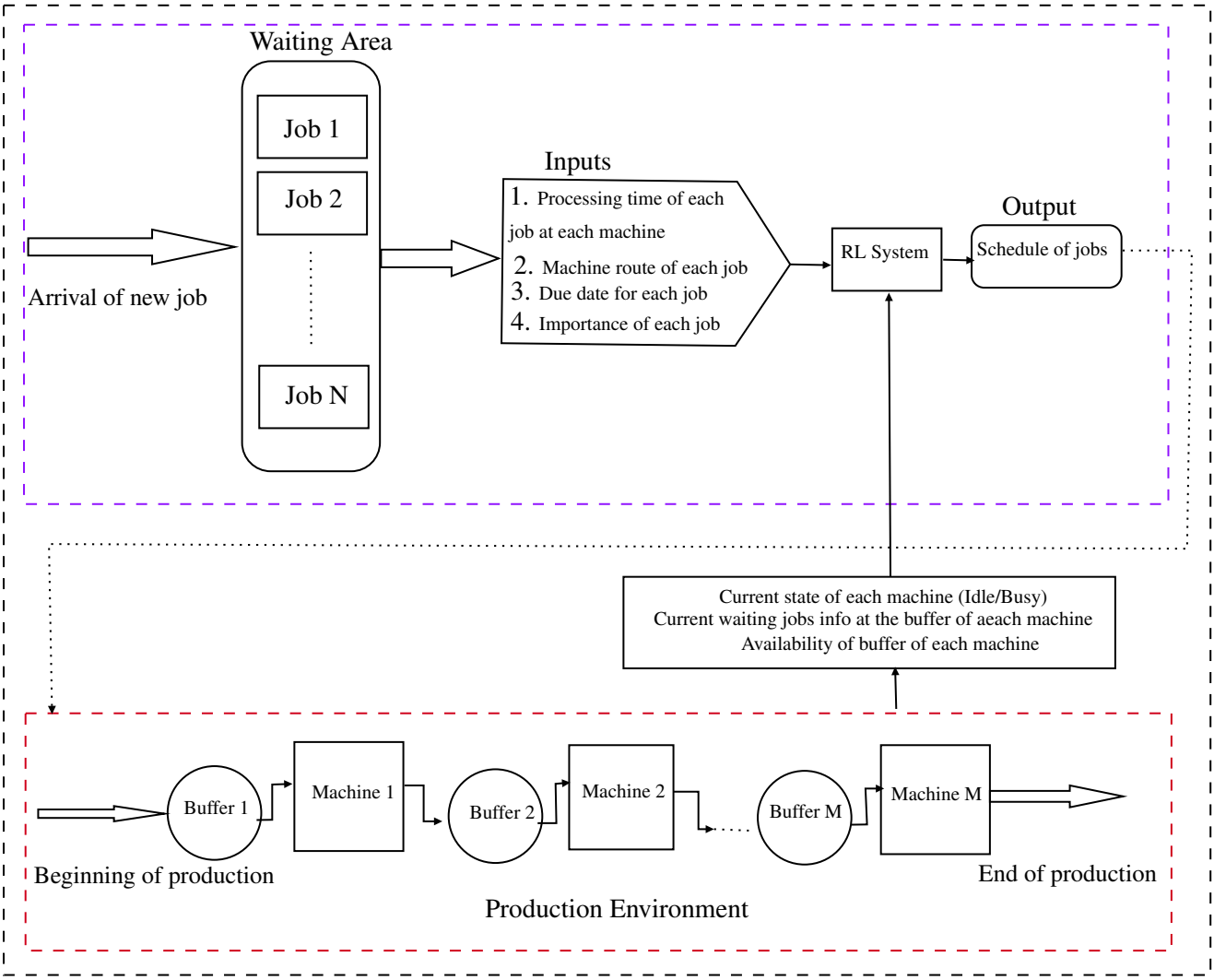

Figure 1: Flow diagram of dynamic job shop production environment.

Where θ_i are the parameters (weights of neural netowrk) of the Q-network at iteration i. The following loss function is used to update the Q-network.

$$L_i(\theta_i) = E_{(s,a,s',r)}\ U(D)[(r+\gamma \max_a Q(s',a';\theta_i^-) - Q(s,a,;\theta_i))^2] \tag{1}$$

To satisfy the stability and divergence of the function, DQN uses experience replay and fixed Q-targets:

- **Experience Replay**: is used to generate uncorrelated experiences data for online training of deep RL agents. It separates the learning phase and gaining experience. The discovered experiences $((s_t, a_t, s_{t+1}, r_t))$ are stored in a large table and then based on taking random samples uniformly $(U(D))$ from this table, learning is performed. The main advantage of this approach, it provides better convergence when training the DQN. On the other hand, to make experience replay more efficient, Schaul et al. (2015) proposed prioritized experience replay. The key idea is that some transitions can increase the learning performance more than from others.
- **Fixed Q-Targets**: In deep Q learning, to calculate the loss (aka TD (Temporal Difference) error), we calculate the difference between the TD target and current Q value. But we do not know the real

TD target. So using the same parameters for estimating the target and Q-value, cause the correlation between them. DQN uses a separate network with a fixed parameter called θ_i^- for estimating the TD target.

The flow diagram of DQN for our problem is illustrated in Figure 2. The pseudo-code of the DQN algorithm for our problem is given in Figure 3.

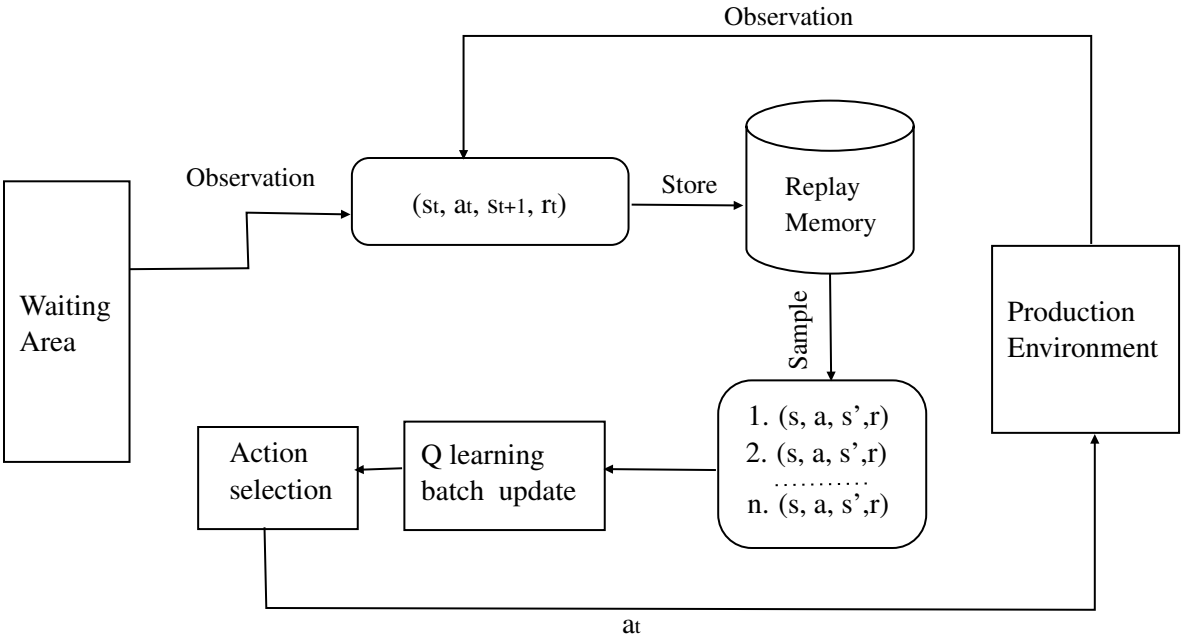

Figure 2: Flow diagram of DQN for DJSP.

3.1.1 State Space

The next task selection depends on a lot of factors. These factors are given in Figure 1. The state space of the learning problem should be defined based on these factors. The state space of the environment is represented as follows:

i: Jobs
j: Machines/Buffers
X_i: [Arrival Time, Processing Time, Machine Route, Due date, Importance]
Y_j: [Current state (Idle/Busy), availability of buffer, current waiting jobs at the buffer]

These features makes state space too large to use tabular learning methods. So methods based function approximation seems quite appropriate to estimate the value function of each state. In this study, DQN (Deep Q-learning) is used as a function approximation method.

3.1.2 Action Space

In DQN, the intelligent agent needs to decide which job should be selected from the waiting jobs and should send it to its first machine route. Also, it should determine its sequence among the jobs waiting on the buffers of machines.

Algorithm 1: DQN pseudocode for DJSP

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function Q^- with weights $\theta = \theta^-$
Fitness value evaluation for each individual
for *Episode 1:n* **do**
  for *t=1,T* **do**
    With probability ε select a random action a_t
    otherwise select a_t=arg min$_a Q(s_t, a, \theta)$
    Execute action a_t, observe reward r_t and state s_{t+1}
    Store (s_t, a_t, r_t, a_{t+1}) into the replay memory (E)
    Randomly select a minibatch of experiences from E

$$y_j = \begin{cases} r_j, & \text{if it is a terminal state} \\ r_j + \arg\min_a Q(s_t, a, \theta^-), & \text{otherwise} \end{cases}$$

    Perform a gradient descent step on DNN loss function $(y_j - Q(s_j, a_j, \theta))^2$ with respect to θ
    Every C steps set $\theta^- \leftarrow \theta$
  end
end

Figure 3: DQN pseudocode for DJSP.

3.1.3 Reward Mechanism

The main purpose of the reward mechanism is to help to reduce the cost because of the selection of wrong actions. In literature, some performance measures such as makespan, mean lateness, number of tardy jobs are considered as objective to be minimized. This performance measure is very useful in static job scheduling problems. But in dynamic job scheduling problems, to generate a schedule based on one of these factors may not be profitable in the long term due to uncertainty. So the delay time of the jobs can be considered as the main performance measure. On the other hand, the jobs may not be equally important. For example, the agent may decide to feed a new arrival job into the buffer and if there was no available space in the buffer, the job with the least priority can be transferred to the first waiting area to provide space for the new job. But this transfer causes some costs. Moreover, the capacity of the buffers of machines is considered when evaluating the reward mechanism. So the reward mechanism is defined as follows:

$$R = \sum_i I_i * D_i + \sum_i T_i * J_i + \sum_j K_j * (E_j) \tag{2}$$

In equation (1), I_i and D_i represents the cost of one unit time delay of job i and delay time of job i after taking an action. E_j is a binary variable, it takes 1 if the buffer capacity of machine j is exceeded after the action is taken, otherwise, it equals zero. K_j is a parameter which represents the cost of excessing buffer capacity at machine j. T_i is a binary variable, it takes 1 if the job i is transferred to the waiting area when it waits on the buffer of a machine. J_i is a parameter. In fact, the Equation (2) indicates a cost function which needs to be minimized by taking the correct actions.

4 EXPERIMENT RESULTS AND ANALYSIS

4.1 Training Configuration

A hypothetical dynamic JSP is considered in this study. We have assumed that there are 10 unique jobs with their special machine routes and operation times on each of 5 machines. Each job arriving at the system belongs to one of these job types based on uniform distribution. The time between job arrivals to the system follows an exponential distribution with a mean of 12. The processing times of jobs were uniformly distributed between 6 and 8. The due date of each job is unique regardless of its type. The discrete event simulation approach is adopted to train a DQN agent under the following assumptions.

- A job cannot be divided. Therefore, two or more operations of the same job cannot be processed at the same time.
- Technical precedence order of jobs can not be violated.
- Jobs cannot be cancelled. Each job should be processed until it is completed.
- Machine failures have been ignored.
- The time required to move parts between workstations is also ignored.
- The length of queues in front of machines are limited to their buffer size.
- Even if a job is queued in front of the machine, it may change its position in the queue or be sent back to the waiting area.

The algorithm is implemented in python. Keras library is used to build neural networks. The hyper-parameters of the DQN algorithm, which are determined by the trial and error approach, are shown in Table 1.

Table 1: The hyper parameters of DQN algorithm.

Hyper-parameter	DQN
Replay Buffer Size	10000
Batch Size	28
Discount factor	0.95
Learning Rate	0.007

4.2 Experiment Results

The DQN agent was trained under the above experiment conditions and the average delayed time and unit scheduling costs of the finished jobs are shown in Figure 4 and Figure 5, respectively. The training process was allowed to continue until 100,000 jobs were processed through all machines.

Figure 4 shows the average delay time of finished jobs during the training process. In the early period of training, the agent selects the actions randomly to store in Replay Memory. After a certain time for collecting enough experiment, the optimization effects of learning phase comes out. When the training is performed to some extent, the average delay time of jobs no longer decreases significantly, but fluctuates smoothly, which implies that the agent has learned a stable control strategy. Figure 5 shows the unit scheduling cost of finished jobs (i.e., the average scheduling cost of finished jobs). It follows a similar pattern to Figure 4 in the training process.

4.2.1 Model Comparison

The dispatching rules are commonly used for dynamics job scheduling problems since they are simple and good fixed-policies. The weakness of these rules is that they are only able to handle one performance measure. For instance, Earliest Due Date (EDD) was shown to best for minimizing the maximum tardiness

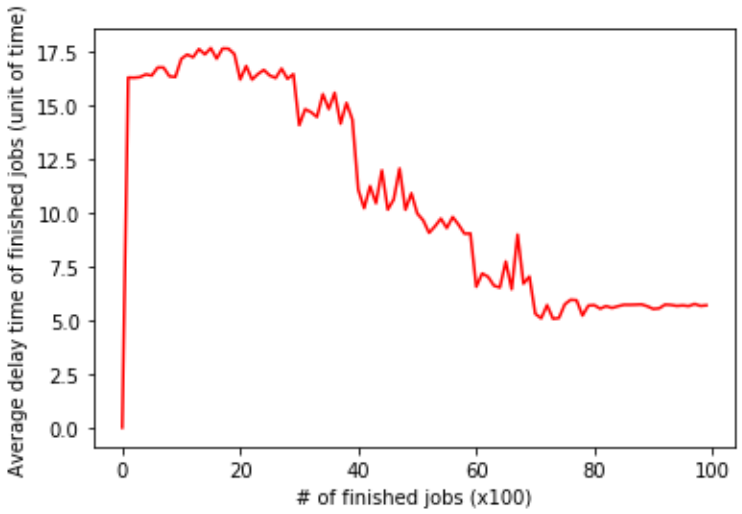

Figure 4: The training process of the agent for average delay time.

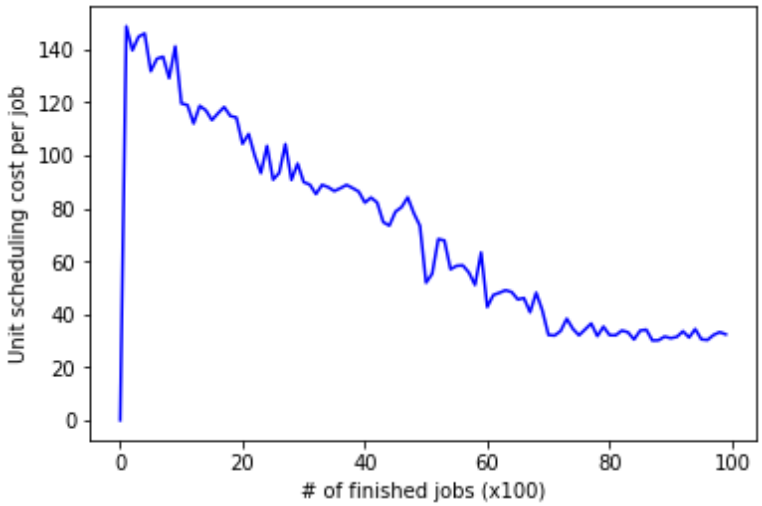

Figure 5: The training process of the agent for unit scheduling cost.

(Barman 1998). To test the performance of DQN model, we compared the results of it with following dispatching rules:

- Shortest Processing Time (SPT) : Highest priority is given to the job having the shortest processing time
- Earliest Due Date (EDD): Highest priority is given to the job having earliest due date.

These rules were selected due to their success on due-date based objectives (Chiang and Fu 2007). This is also in line with our reward function. We have generated 1000 jobs with the same training configuration (see Section 4.1). The generated jobs were scheduled by using the training DQN model and two dispatching rules. Figure 6 shows that the DQN model outperforms than two dispatching rules in the long run.

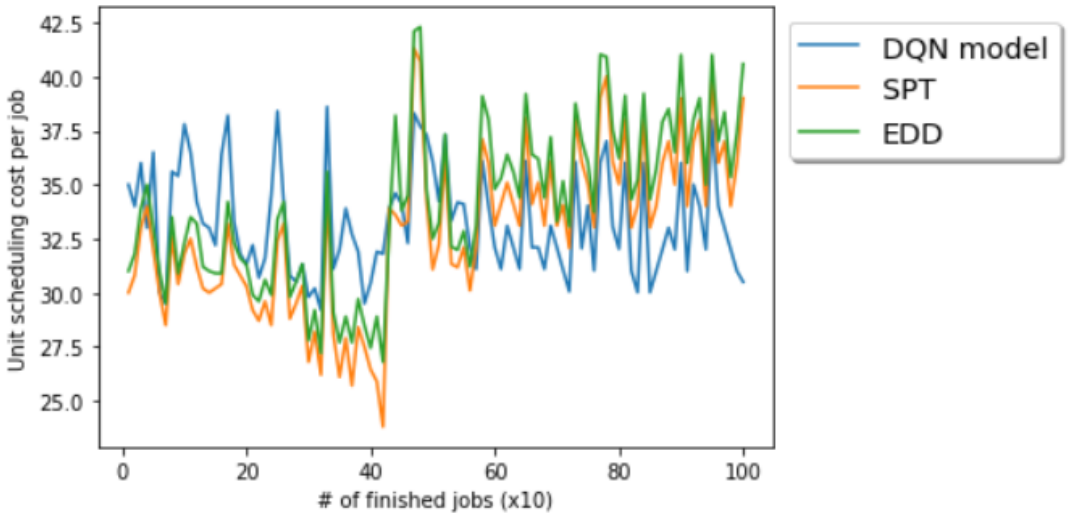

Figure 6: Comparison of DQN model with two heuristics.

5 CONCLUSION

In this study, the dynamic job scheduling problem, which consists of n work and m machines, and where the jobs arrive at the system continuously, is considered. We have illustrated how the DQN algorithm can be adopted to solve this dynamic JSP based on discrete-event simulation. The proposed approach radically improves the scheduling cost and delay time of jobs according to the initial position. The model performance was further tested by comparing it with two common dispatching rules. The results are promising when considering the long-term outcome. The foundations of this study provide a basis for further studies. Our future work will aim at exploiting the new advancements of RL researches for obtaining better solutions to the dynamic JSP. Moreover, we will investigate the other dynamic events, such as cancellation of the order, machine breakdowns.

REFERENCES

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

Barman, S. 1998. "The Impact of Priority Rule Combinations on Lateness and Tardiness". *Iie Transactions* 30(5):495–504.

Baykasoğlu, A., and F. S. Karaslan. 2017. "Solving Comprehensive Dynamic Job Shop Scheduling Problem by Using a GRASP-based Approach". *International Journal of Production Research* 55(11):3308–3325.

Branke, J., and C. W. Pickardt. 2011. "Evolutionary Search for Difficult Problem Instances to Support the Design of Job Shop Dispatching Rules". *European Journal of Operational Research* 212(1):22–32.

Chen, B., and T. I. Matis. 2013. "A Flexible Dispatching Rule for Minimizing Tardiness in Job Shop Scheduling". *International Journal of Production Economics* 141(1):360–365.

Cheng, R., M. Gen, and Y. Tsujimura. 1996. "A Tutorial Survey of Job-shop Scheduling Problems Using Genetic Algorithms—I. Representation". *Computers & industrial engineering* 30(4):983–997.

Chiang, T.-C., and L.-C. Fu. 2007. "Using Dispatching Rules for Job Shop Scheduling with Due Date-based Objectives". *International journal of production research* 45(14):3245–3262.

Fang, J., and Y. Xi. 1997. "A Rolling Horizon Job Shop Rescheduling Strategy in the Dynamic Environment". *The International Journal of Advanced Manufacturing Technology* 13(3):227–232.

Gabel, T., and M. Riedmiller. 2008. "Adaptive Reactive Job-shop Scheduling with Reinforcement learning agents". *International Journal of Information Technology and Intelligent Computing* 24(4):14–18.

Gabel, T., and M. Riedmiller. 2012. "Distributed Policy Search Reinforcement Learning for Job-shop Scheduling Tasks". *International Journal of production research* 50(1):41–61.

Garey, M. R., D. S. Johnson, and R. Sethi. 1976. "The Complexity of Flowshop and Jobshop Scheduling". *Mathematics of operations research* 1(2):117–129.

Gere Jr, W. S. 1966. "Heuristics in Job Shop Scheduling". *Management Science* 13(3):167–190.

Kiran, A. S., and M. L. Smith. 1984. "Simulation Studies in Job Shop Sheduling: A Survey". *Computers & Industrial Engineering* 8(2):87–93.

Larsen, R., and M. Pranzo. 2019. "A Framework for Dynamic Rescheduling Problems". *International Journal of Production Research* 57(1):16–33.

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski et al. 2015. "Human-level Control Through Deep Reinforcement Learning". *Nature* 518(7540):529–533.

Mohan, J., K. Lanka, and A. N. Rao. 2019. "A Review of Dynamic Job Shop Scheduling Techniques". *Procedia Manufacturing* 30:34–39.

Morton, T., and D. W. Pentico. 1993. *Heuristic Scheduling Systems: with Applications to Production Systems and Project Management*, Volume 3. John Wiley & Sons.

Ramasesh, R. 1990. "Dynamic Job Shop Scheduling: A Survey of Simulation Research". *Omega* 18(1):43–57.

Riedmiller, S., and M. Riedmiller. 1999. "A Neural Reinforcement Learning Approach to Learn Local Dispatching Policies in Production Scheduling". In *IJCAI*, Volume 2, 764–771. New York: Association for Computing Machinery Digital Library.

Schaul, T., J. Quan, I. Antonoglou, and D. Silver. 2015. "Prioritized Experience Replay". *arXiv preprint arXiv:1511.05952*.

Suwa, H., and H. Sandoh. 2007. "Capability of Cumulative Delay Based Reactive Scheduling for Job Shops with Machine Breakdowns". *Computers & Industrial Engineering* 53(1):63–78.

Turker, A. K., A. Aktepe, A. F. Inal, O. O. Ersoz, G. S. Das, and B. Birgoren. 2019. "A Decision Support System for Dynamic Job-shop Scheduling Using Real-time Data with Simulation". *Mathematics* 7(3):278.

Wang, Y.-C., and J. M. Usher. 2005. "Application of Reinforcement Learning for Agent-based Production Scheduling". *Engineering Applications of Artificial Intelligence* 18(1):73–82.

Xiong, H., H. Fan, G. Jiang, and G. Li. 2017. "A Simulation-based Study of Dispatching Rules in a Dynamic Job Shop Scheduling Problem with Batch Release and Extended Technical Precedence Constraints". *European Journal of Operational Research* 257(1):13–24.

Zhang, T., S. Xie, and O. Rose. 2017. "Real-time Job Shop Scheduling Based on Simulation and Markov Decision Processes". In *2017 Winter Simulation Conference (WSC)*, edited by W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, 3899–3907. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Zhang, W., and T. G. Dietterich. 1995. "A Reinforcement Learning Approach to Job-shop Scheduling". In *IJCAI*, Volume 95, 1114–1120. Citeseer.

AUTHOR BIOGRAPHIES

YAKUP TURGUT is a PH.D. candidate in industrial engineering at Istanbul Technical University. He received a B.E. degree in industrial engineering from Yıldız Technical University, Turkey, in 2015, and a master's degree from the Istanbul Technical University, Turkey, in 2018. His research interests are simulation modeling, mathematical modeling, and artificial intelligence. His email address is turgut16@itu.edu.tr.

CAFER ERHAN BOZDAG is an Assistant Professor in the industrial engineering department at Istanbul Technical University. He received the B.E., M.E., and Ph.D. degrees from Istanbul Technical University, Istanbul, Turkey. His research interests focus on system analysis, system simulation, agent-based simulation, system dynamics, and fuzzy logic. His e-mail address is bozdagc@itu.edu.tr.