Proceedings of the 2024 Winter Simulation Conference
H. Lam, E. Azar, D. Batur, S. Gao, W. Xie, S. R. Hunter, and M. D. Rossetti, eds.

OPTIMIZING JOB SHOP SCHEDULING PROBLEM THROUGH DEEP REINFORCEMENT LEARNING AND DISCRETE EVENT SIMULATION

Bulent Soykan[1], and Ghaith Rabadi[1]

[1]School of Modeling, Simulation, and Training, University of Central Florida, Orlando, FL, USA

ABSTRACT

This paper explores the optimization of Job Shop Scheduling Problem (JSSP), by employing a Deep Reinforcement Learning (DRL) approach that learns optimal scheduling policies through continuous interaction with a job shop setting simulated within a discrete event simulation (DES) environment. The study involves computational experiments to train and evaluate the DRL agent across benchmark JSSP instances. We compare our DRL-based scheduling solutions against generated by widely used priority dispatching rules, assessing the impact of the learned policies on performance metrics including the total time to complete all jobs (makespan) and efficiency in using machines (machine utilization). Our results indicate improvements in scheduling efficiency, showcasing the DRL algorithm's ability to adapt and optimize in complex scheduling scenarios. This paper underscores the potential of integrating DRL with DES to create a powerful toolset for modern manufacturing challenges, enabling businesses to maintain competitive advantage through improved operational agility.

1 INTRODUCTION

The modern manufacturing landscape requires not only efficiency but also a significant degree of adaptability and responsiveness to remain competitive in the face of quickly changing market conditions. To address these evolving requirements, a shift towards exploring and understanding the Job Shop Scheduling Problem (JSSP) becomes imperative. This real-world problem has developed over decades, highlighting the need for adaptive and dynamic scheduling solutions (Xiong et al. 2022). Presently, the primary goal is to devise scheduling systems that are both flexible and capable of swiftly responding to unpredictable market dynamics and production demands. The complexities of the JSSP have been thoroughly explored by both academic researchers and industry professionals, who recognize its essential role in enhancing operational agility, minimizing downtime, and ultimately, contributing to greater cost efficiency. As the industry progresses, there emerges a growing demand for systems that are not only adaptable to changing scenarios but also supportive of real-time decision-making. This need for innovative approaches propels the advancement of dynamic and adaptive job shop scheduling solutions tailored to address the modern manufacturing challenges efficiently.

The classic JSSP is a widely recognized combinatorial optimization problem and it involves optimizing the allocation of resources to tasks over time. Specifically, the JSSP requires determining the optimal schedule for processing a set number of given jobs on a set number of machines simultaneously while adhering to specific constraints (sequence-dependent setup times, machine availability, and job priorities etc.) and objectives (minimize the makespan, tardiness, maximum lateness etc.). Due to its combinatorial nature and NP-hard computational complexity, JSSP is a well-established problem in operations research (OR) and computer science literature. Traditional methods such as heuristic approaches, mathematical programming and meta-heuristic methods have been used to solve this problem. However, as the landscape of real-world operations becomes increasingly dynamic and susceptible to uncertainties, these traditional methods experience limitations in addressing the challenges presented by the dynamic optimization of the JSSP. In essence, this requirement necessitates a novel perspective that can harness the power of cutting-edge technologies (Manne 1960). Recently, significant progress in artificial intelligence and machine learning (ML), particularly in reinforcement learning (RL) has delivered promising outcomes in various combinatorial optimization problems.

Real-world combinatorial optimization problems, including the JSSP, are characterized by vast state spaces that render the problem computationally challenging. To tackle the intricate complexities of JSSP, we present a Deep Reinforcement Learning (DRL)-based approach to dynamically adjust scheduling in response to evolving conditions. We also developed a discrete event simulation (DES) that closely mirrors the dynamics of a job shop scheduling setting. Our research objective is to enhance adaptive scheduling and operational flexibility. Such improvements hold the promise of not only reducing costs and inefficiencies but also significantly increasing the responsiveness and robustness of manufacturing operations.

The primary research questions addressed in this study are: To what extent can a DRL agent learn adaptive scheduling strategies through trial-and-error interactions with the DES of JSSP? Does DES representation enable the exploration of various JSSP scenarios, creating a rich training environment for the DRL agent? This paper makes several scientific contributions to the field of sequential decision-making and simulation modeling. Firstly, we introduce an approach that integrates DRL with DES for the JSSP. This contribution is significant for advancing our understanding of dynamic scheduling processes and offering a transformative solution to JSSP instances. Secondly, we develop a specialized algorithm within the DRL framework that learns Priority Dispatching Rules (PDRs). This aspect of our work addresses the challenge of operational inefficiency in job shops. Additionally, the design and implementation of the DES provides a depiction of the job shop scheduling process. This aspect of our research is important in creating a detailed and realistic training environment for the DRL agent, thereby enabling it to encounter and learn from a variety of JSSP scenarios. The simulation model not only facilitates a deeper understanding of the dynamics in job shop scheduling but also significantly enhances the agent's ability to devise robust scheduling policies that can readily adapt to varying conditions.

The remainder of the paper is organized as follows. Section 2 presents the background information regarding the JSSP and DRL, and provides a summary of our literature review. Section 3 is dedicated to outlining the methodology, including the detailed problem definition, development of our DES environment and the DRL framework. Section 4 provides the details of the DRL agent training process and experimental results. Finally, Section 5 concludes the paper and highlights future work directions.

2 BACKGROUND AND RELATED WORK

2.1 Job Shop Scheduling Problem

The JSSP seeks to determine an optimal scheduling of a set of jobs to be processed on a limited number of machines, with the primary aim of optimizing one or more objectives such as minimizing the total time to complete all jobs (makespan), minimizing delays beyond due dates (tardiness), or maximizing efficiency in using machines (machine utilization), the promptness of job completion (on-time delivery rates). The significance of JSSP lies in its direct applicability to real-world manufacturing scenarios where efficient utilization of resources directly impacts operational costs, delivery times, and overall productivity. Formally, the JSSP can be defined as follows: Given a set of n jobs $\{J_1, J_2, ..., J_n\}$ that need to be processed on a set of m machines $\{M_1, M_2, ..., M_m\}$, each job J_i consists of an ordered sequence of operations $\{O_{i1}, O_{i2}, ..., O_{im}\}$, with each operation O_{ij} requiring processing on machine M_j for a specified duration p_{ij}. The goal is to find a schedule, i.e., a sequence of job operations on each machine, that optimizes the given objective(s), subject to the constraints that each machine is capable of handling a single operation at a time, each job operation needs to be completed before the next one in the job sequence can begin, and once an operation starts on a machine, it must run to completion without interruption.

The JSSP have extensively researched and these research efforts have evolved from deep mathematical analysis to using simpler rules-based heuristic methods and more complex meta-heuristic strategies. Initially, solving JSSP started with mathematical methods like linear programming, focusing on finding the most effective job sequences under various limitations (Dantzig and Wolfe 1960). However, as the problems grew in size, these exact mathematical solutions became impractical because they required too much computing power for larger job shop setups. This limitation led to a shift towards heuristic approaches. Heuristic methods, such as applying PDRs like First In, First Out (FIFO) or Shortest Processing Time (SPT), aimed to find good solutions more simply and quickly, making them more applicable to real-world scenarios despite their inability to guarantee the optimal solution (Holthaus and Rajendran 1997). The approach to solving JSSP further advanced with the introduction of meta-heuristic algorithms like Genetic Algorithms and Simulated Annealing. These algorithms are designed to look through possible solutions more thoroughly than the simpler heuristics, attempting to find a balance between exploring new solutions (exploration) and making the most of known good solutions (exploitation) (Cheng, Gen, and Tsujimura 1996). Most recently, the integration of ML, particularly DRL, has begun to provide more dynamic and intelligent scheduling solutions, reflecting an increased capacity to tackle the JSSP more effectively and adaptively (Zhang et al. 2020).

2.2 Deep Reinforcement Learning

RL is a sub-field of ML where an agent acquires decision-making skills by executing actions and evaluating the results through a process of trial and error (Sutton and Barto 2018). The core idea of RL is for an agent to take actions within a defined space, receive feedback in the form of rewards or penalties, and use this feedback to learn optimal behavior or decision-making strategies over time. A fundamental algorithm in traditional RL is Q-learning, which enables an agent to learn the value of taking certain actions in specific states, thereby guiding the development of optimal policies over time without requiring a model of the environment. It serves as a key method upon which many RL strategies are built upon (Watkins and Dayan 1992). This field has seen substantial growth with the advent of deep learning, leading to the inception of DRL. DRL combines the decision-making framework of traditional RL with the powerful feature extraction capabilities of deep neural networks (DNNs) (Arulkumaran et al. 2017). This convergence enables agents to learn optimal strategies in environments characterized by high-dimensional state spaces, which were previously infeasible with classic RL approaches alone. DRL algorithms can be broadly classified into value-based and policy-based. Value-based DRL, such as Deep Q-Learning, focuses on estimating the expected total reward for each action in a given state. On the other hand, policy-based DRL algorithms aim to learn and optimize the policy directly (Sutton, Singh, and McAllester 2000).

DRL has demonstrated remarkable success across various domains. In gaming, agents trained via DRL have achieved human-level performance in complex games like Go and various Atari games, showcasing their strategic depth and decision-making abilities (Mnih et al. 2013). In robotics, DRL has been used to teach robots to perform tasks such as walking, grasping, and cooperative manipulation with precision and adaptability (Gu et al. 2017). Also, in autonomous vehicles, it contributes to developing sophisticated decision-making algorithms that help navigate complex and dynamic road environments safely (Kiran et al. 2021). These successes underscore the versatility and potential of DRL in tackling problems that require nuanced understanding and interaction with complex environments.

In 2013, the DeepMind team revolutionized DRL with the introduction of Deep Q-Learning, employing Deep Q-Networks (DQNs) that use a Convolutional Neural Network (CNN) trained through Q-Learning (Mnih et al. 2013). This method involves interpreting game states as inputs and calculating the quality (Q-values) of potential actions, updating the network via replay memory, and a combination of immediate rewards and anticipated future rewards. Also, the introduction of a secondary, periodically updated Target network enhances stability by mitigating positive feedback loops that could lead to sub-optimal learning, while a ε-greedy exploration strategy balances between random actions and those suggested by the Q-network to ensure continual learning progress (Mnih et al. 2015).

2.3 Integrated Studies

The integration of DRL with DES models has seen an encouraging trend in recent years across various domains of optimization problems. The fusion of DRL's adaptive learning capabilities with the dynamic representation power of DES offers a potent combination for tackling complex decision-making problems by addressing the challenge of real-world uncertainties and reducing the need for extensive real-life trials, thus making the DRL agent training process more efficient and cost-effective (Belsare, Badilla, and Dehghanimohammadabadi 2022). Notable studies have demonstrated this synergy in industrial scheduling problems by utilizing the potential of DRL to learn optimal policies from the rich DES environments mirroring the complexity and variability of real-world systems. (Zhang and Dietterich 1995) present the pioneering application of RL for production scheduling, illustrating their capability to swiftly generate high-quality solutions with the aim of future advancements minimizing domain-specific engineering requirements, thereby paving the way for cost-effective scheduling solutions.

Subsequent studies have explored the integration of DRL with DES and other technological advancements to address specific scheduling problems. (Waschneck et al. 2018) propose the use of DQN agents for the autonomous optimization of multiple production targets within a semiconductor production scheduling framework. Utilizing a MATLAB factory simulation as a DES tool, this study highlights the versatility of DRL by comparing its performance against standard PDR heuristics and showcasing efficiency improvements across various industrial settings. (Han and Yang 2020) introduce a tailored DRL approach that employs disjunctive graph dispatching alongside CNNs to address the limitations of traditional solution methods in complex production environments. The proposed approach, which includes the use of the dueling double DQN with prioritized replay, demonstrates its efficacy through static computational experiments on several JSSP instances, showing superior performance especially in dynamic environments with variable processing times. (Lang et al. 2020) demonstrates the efficacy of DQN in deriving high-quality, real-time scheduling solutions for a flexible job shop with integrated process planning, underscoring DQN's low training requirements and its agents' capacity to generalize solutions to new problems.

In a recent review, (Panzer and Bender 2022) assess DRL's performance across various domains including process control and maintenance. This study acknowledges DRL's ability to surpass conventional algorithms by managing uncertainties more adeptly, though it also calls for further research to enhance the practical application and optimization of DRL in production systems. (Hammami et al. 2022) address JSSP with real-time job arrivals using Proximal Policy Optimization Actor-Critic and Graph Neural Networks (GNNs). This study highlights the continuous adaptability and potential for efficiency gains through real-time algorithmic adjustments, albeit noting the extended training required for optimal policy convergence. (Devanga, Badilla, and Dehghanimohammadabadi 2022) illustrate the effective combination of DRL and DES to address complex scheduling problems within industrial settings, utilizing Simio DES software and Python. Their methodology underscores the potential of embedding AI-driven operations in simulation environments, thereby advancing decision-making capabilities.

2.4 Literature Review Summary

Despite these successes, existing research efforts reveal gaps. One common limitation is the difficulty in accurately reflecting the dynamics of real-world manufacturing environments in the optimization process. Moreover, many studies focus on isolated aspects of the scheduling problem, such as sequencing or timing, without a comprehensive approach that captures the entire spectrum of operational decisions, including machine allocation and job prioritization. Additionally, there's a notable scarcity in research that addresses the dynamic nature of manufacturing systems.

Our research aims to bridge these gaps by proposing an approach that leverages DRL with a DES for JSSP. By creating a DES environment that represents the dynamics and complexities of real-world manufacturing processes, the DRL agent is trained to identify and adapt to patterns, leading to more robust and adaptive scheduling solutions. Unlike existing studies, our approach encapsulate both assignment and sequencing decisions, providing a holistic optimization strategy that can dynamically adjust to changes and disruptions. The hypothesis underpinning this research is that the combination of DRL and DES holds the potential to improve the efficiency of JSSP. This improvement comes from the DRL agent's ability to learn complex, nuanced strategies for resource allocation and task prioritization that are beyond the scope of traditional heuristics. Moreover, the DES provides a safe and scalable environment for the agent to explore a wide range of scheduling scenarios, including those involving uncertainties and operational changes, thereby enhancing the adaptiveness of scheduling solutions.

3 METHODOLOGY

3.1 Discrete Event Simulation Environment

The creation of a simulation environment is one of the important steps for solving JSSP with DRL. The simulation environment acts as a virtual sandbox where a DRL agent can safely explore and learn, encountering a variety of scheduling scenarios reflective of real-world complexities. Recently there has been a growing interest on creating simulated environments that integrate DES with RL experimentation modules/tools, aiming to supply DRL agents with dynamic contexts for mastering adaptive scheduling techniques while serving as platforms for both training and assessing their adaptability and problem-solving capabilities under varied operational conditions. (Lang et al. 2021) introduce a methodology that transforms the complex challenges of production scheduling into RL-compatible models using DES with OpenAI Gym interface, facilitating the development, deployment, and refinement of RL algorithms for job allocation and sequencing; this allows RL agents to engage directly with complex simulations, simplifying intricate scheduling tasks into manageable decisions. While their framework demonstrates the potential for customizing DES for DRL training and offers insights into RL agents' learning processes, it raises limitations about compatibility with DES tools not supporting Python, the primary language for OpenAI Gym and many ML libraries.

We designed and developed a DES, tailored for the JSSP using SimPy Python library. This simulation model aims to mimic real-world manufacturing environments, where jobs with varying operations need to be allocated to specific machines with the goal of optimizing key performance indicators (KPIs). Also, it has an OpenAI Gym-like interface which enables the application of DRL algorithms to learn and propose optimal scheduling policies. The simulation begins with reading the job information from an Excel file, which contains the specifics of each job that needs to be scheduled, including process order and duration on different machines. This setup allows for a customizable and realistic simulation scenario reflective of actual job shop operations. The core of the simulation is the "Factory" module, which models the job shop environment. This environment includes a set of machines and a queue of jobs awaiting processing according to certain PDRs, namely FIFO, SPT, Last In First Out (LIFO), Longest Processing Time (LPT), Least Total Work Remaining (LTWR), Most Total Work Remaining (MTWR), Shortest Total Processing Time (STPT), Longest Total Processing Time (LTPT), defined in the "Action Space" module.

During the simulation, multiple runs (replicates) are conducted to gather statistical significance in the outcomes. In each run, the simulation environment is reset, and a DRL agent interacts with this environment by choosing actions depending on the current state of the job shop. The environment then progresses to a new state, providing the agent with rewards based on the efficiency of the chosen action. This iterative process continues until the completion of all jobs. Upon the conclusion of each run, performance metrics, namely total rewards, makespan and average machine utilization are calculated and recorded. This simulation setting not only enables the analysis of job shop scheduling policies but also facilitates the training of DRL agents in an environment that accurately reflects the complexities and uncertainties of real-world manufacturing processes. The use of a familiar OpenAI Gym interface ensures that the simulation can be easily integrated with existing DRL algorithms, making it a valuable tool for both research and practical applications in optimizing the JSSP.

3.2 Deep Reinforcement Learning Framework

The application of DRL to the JSSP involves formulating the problem as a sequential decision-making problem under uncertainty. Therefore, we framed the problem as a Markov Decision Process (MDP). This formulation is essential for addressing the dynamic nature of the problem where an agent can learn to make scheduling decisions through interactions with the environment. An MDP fundamentally serves as a mathematical framework for making decisions in a sequence where results are partially influenced by chance and partially under the control of a decision-maker. The MDP framework for the JSSP is defined by the tuple $M = (S, A, R, T, \gamma, H)$, where:

- S is the state space representing the current configuration of the job shop, including information about the jobs, their respective stages of completion, and the status of each machine. Each state $s \in S$ encapsulates the necessary details to make informed scheduling decisions, namely, the queue of pending operations and the operational status of machines.
- A is the action space that consists of all possible scheduling actions an agent can take at any given state. In the context of JSSP, an action $a \in A$ represents the selection of a specific job to be assigned to a particular machine next, based on the PDRs defined in the DES.
- $R : S \times A \times S \to \mathbb{R}$ is the reward function that assigns a numerical reward to transitions between states as a result of an action. R is designed to encourage the reduction of the makespan and improvement in machine utilization. The agent receives a positive reward for actions that lead to a decrease in the overall completion time of all jobs and increase the utilization of machines.
- $T : S \times A \to S$ represents the transition function which defines the probabilistic outcome of taking action a in state s, resulting in transitioning to a new state s'. Since the scenarios are stochastic, it provides probabilities for reaching various subsequent states.
- $\gamma \in [0, 1]$ is the discount factor which establishes the current value of future rewards. A lower value of γ makes the agent prioritize immediate rewards over distant future rewards, while a higher γ encourages the agent to consider long-term benefits.
- H specifies the horizon of the decision process and represents the maximum length of each episode.

The MDP framework for the JSSP, as defined above, provides a structured approach for applying DRL algorithms to learn optimal or near-optimal scheduling policies. By iteratively interacting with the job shop environment modeled as an MDP, the DRL agent explores the state-action space and accumulate experience, which guides the development of a strategy that maximizes the cumulative rewards - essentially learning to efficiently schedule jobs with the aim of improving KPIs within the manufacturing process.

3.3 Double Deep Q-Network Algorithm

At the core of the DQN algorithm lies the capability of DNNs to serve as universal function approximators. This feature enables the DQN to effectively estimate the Q-function, which is essential for mapping a given state to the Q-values of all possible actions from that state. The learning process of a DQN involves initializing Q-value estimates and refining these estimates over time through interaction with the environment and adherence to an ε-greedy policy. The architecture of DQN comprises two key components: the Q network, which learns to predict the optimal state-action values, and the Target network, which assists in stabilizing the learning updates. Additionally, the DQN utilizes an Experience Replay mechanism, which facilitates the collection and utilization of past experience data for training the Q network, thereby enhancing the efficiency and effectiveness of the learning process.

Building on the foundation laid by DQN, the Double Deep Q-Network (DDQN) algorithm introduces an improvement designed to address the challenge of overestimation of action values encountered in DQN. By employing two networks, similar to DQN, DDQN differentiates itself through its update mechanism:

$$Q_{\text{new}}(s_t, a_t) = Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, \arg\max_a Q(s_{t+1}, a; \theta_t); \theta_t^-) - Q(s_t, a_t) \right] \quad (1)$$

where (θ_t) represents the parameters of the current Q-network used for selecting the action, whereas (θ_t^-) are the parameters of the Target network used for evaluating the action, making the DDQN's update mechanism more reliable and less prone to overestimations. This adjustment enables the DDQN to offer a more balanced and accurate evaluation of the action values.

We implemented the DDQN algorithm for learning PDRs and serving as a dispatcher during the inference phase due to its refined ability to evaluate actions more precisely. This precision is critical in job shop environments, where decisions immediately influence operational efficiency and overall performance. The DDQN's approach to minimizing overestimation of Q-values leads to a more stable and reliable learning trajectory, a crucial factor in navigating the complexities of job scheduling. Also, our implementation of the DDQN relies on the effective use of Experience Replay memory, which plays a pivotal role in the learning process. Our implementation of the Experience Replay memory consists of two fundamental components: the "Transition" and "Replay Memory" modules. The Transition module captures and records the experience at each time step, consisting of the current state, action taken, reward received, and the subsequent state, forming the basic data unit for training. The Replay Memory module aggregates and manages a collection of such experiences, allowing the algorithm to sample from this diverse pool of past experiences to train the network. This mechanism not only enriches the learning process by providing a varied set of experiences but also significantly enhances the efficiency of training by breaking the correlation between consecutive training samples, thereby fostering a more robust and effective learning environment.

The initial steps of the DDQN algorithm involve setting up a replay buffer for storing experience tuples, the Q-network with randomly initialized weights, and the Target network, initially a clone of the Q-network. The algorithm then enters a loop over episodes, within which it operates on a step-by-step basis for each time step within an episode. For each step, the algorithm opts between exploring the environment via random actions—facilitated by an ε-greedy policy—and exploiting the knowledge acquired so far to choose actions based on the max Q-value as predicted by the Q-network. After executing the selected action, the outcome (reward and next state) is observed and stored as a transition in the replay buffer. The algorithm periodically samples from this buffer to break the correlation between consecutive training samples, which helps in stabilizing the learning process. A key distinction of DDQN is evident in its strategy for updating the Q-values. It decouples the action selection from the evaluation stage during the computation of the target Q-value. Specifically, the action that maximizes the Q-value for the next state is selected using the current Q-network, but its value is evaluated using the Target network. The algorithm implements periodic updates to the Target network by copying the weights from the current Q-network, ensuring the target values are stable yet gradually adapt to the learned values. The algorithm proceeds iteratively improving the policy until a specified performance threshold is reached or the maximum number of episodes is exhausted (see Algorithm 1) .

Overall, our methodology involves a cycle of RL where the DRL agent iteratively interacts with the DES environment, mirroring manufacturing processes to optimize JSSP. Through this cycle, the agent learns optimal scheduling strategies by making decisions, receiving feedback in the form of rewards, and updating its policy to maximize efficiency and productivity in a simulated real-world setting (Figure 1).

4 COMPUTATIONAL EXPERIMENTS AND RESULTS

The study conducted computational experiments on five small to medium size generated JSSP instances in varying scales, including 6×6, 10×10, 15×15, 20×20, and 30×20 configurations, where the first figure denotes the number of jobs and the second figure the number of machines. The experiments were executed within a Python programming environment, leveraging PyTorch for its deep learning functionalities. This setup ran on a workstation running a Windows operating system, which was powered by an Intel(R) Core(TM) i5-1035G4 CPU and 16 GB of RAM.

Algorithm 1 Double Deep Q-Network (DDQN) Algorithm

```
1: Initialize replay buffer $D$ to capacity $N$
2: Initialize action-value function $Q$ with random weights $\theta$
3: Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
4: Observe and preprocess initial state $s_0$ to $\phi_0 = \phi(s_0)$
5: Initialize target update frequency $C$
6: for episode $= 1, M$ do
7:     Initialize sequence with preprocessed state $\phi_1 = \phi(s_0)$
8:     for t $= 1, T$ do
9:         With probability $\varepsilon$ select a random action $a_t$
10:        otherwise select $a_t = \arg\max_a Q(\phi(s_t), a; \theta)$ using the current Q network
11:        Execute action $a_t$ in emulator and observe reward $r_t$ and next state $s_{t+1}$
12:        Preprocess $s_{t+1}$ to obtain $\phi_{t+1} = \phi(s_{t+1})$
13:        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
14:        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
15:        if $\phi_{j+1}$ is terminal then
16:            Set $y_j = r_j$
17:        else
18:            Set $y_j = r_j + \gamma \hat{Q}(\phi_{j+1}, \arg\max_a Q(\phi_{j+1}, a; \theta); \theta^-)$
19:        end if
20:        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the parameters $\theta$
21:        if $t \mod C == 0$ then
22:            Update the weights of the target network: $\theta^- = \theta$
23:        end if
24:    end for
25:    if mean reward over last $N$ episodes $\geq$ specified threshold then
26:        Break and report success
27:    end if
28: end for
```

4.1 Training Process

The training process of the DRL agent, specifically utilizing the DDQN algorithm for optimizing JSSP, involves a series of steps and iterations to learn effective PDRs. This process is both computationally intensive and data-driven, relying heavily on interactions with the simulated DES environment to incrementally improve decision-making capabilities. Initially, the training begins with the agent exploring the state-action space quite broadly, primarily relying on random selections to gain diverse experiences. This exploration is facilitated by an ε-greedy strategy, where the agent randomly chooses actions with probability ε but follows the learned policy with probability $1-\varepsilon$. The value of ε is gradually decayed over time, allowing the agent to shift from exploration to exploitation of the learned policy. At this stage, the agent's actions are more likely to lead to sub-optimal scheduling decisions, but these experiences are crucial for learning the dynamics of the job shop environment.

A significant challenge encountered during the training is balancing the exploration-exploitation trade-off. Adequate exploration is necessary to prevent the agent from converging prematurely to a sub-optimal policy, yet excessive exploration can hinder learning efficiency and prolong the training process. Another challenge is ensuring the stability of learning, given the high dimensionality of the state space and the variability inherent in the JSSP. The use of experience replay memory and the DDQN architecture, which separates the evaluation of the state-action value from the selection of actions, helps to mitigate these

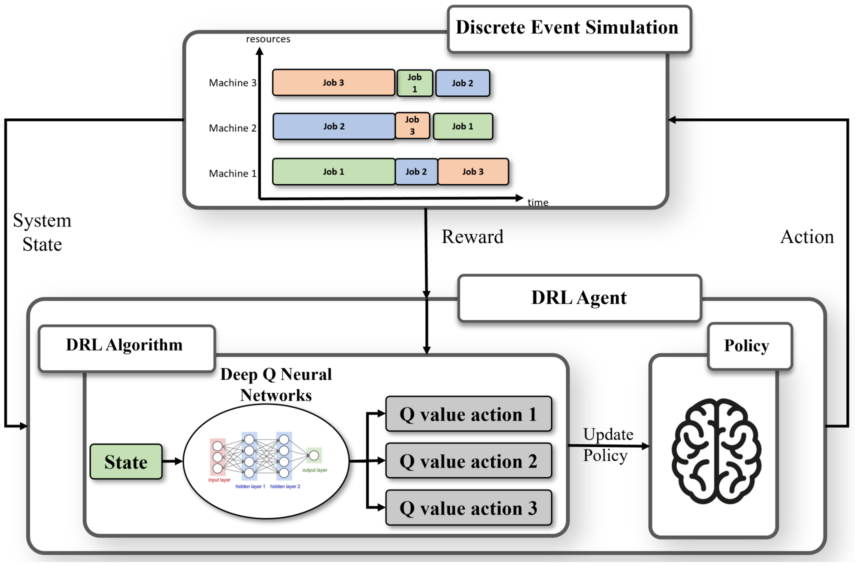

Figure 1: DRL Algorithm and DES.

challenges by providing a more stable learning update mechanism. Throughout the training process, the agent's performance is continuously monitored by evaluating KPIs including makespan and machine utilization. These evaluations provide insight into how well the learned scheduling policy generalizes to unseen instances and identify stages where the agent's performance plateaus or improves.

Towards the later stages of training, as the agent progressively refines its policy, key observations include a noticeable improvement in scheduling efficiency, demonstrated by reduced makespan and increased machine utilization. The agent learns to prioritize jobs and operations in a manner that aligns with the optimization objectives, showcasing the capability of DRL approaches to navigate the complexities and constraints of JSSP. However, continuous monitoring and adjustments are still required to address any overfitting to the training instances and ensure that the agent maintains robust performance across a wide range of job shop configurations.

We use fixed hyperparameters for training, setting the learning rate to 0.001, with a discount factor γ of 0.99 to ensure the agent values future rewards. For exploration, ε starts at 1.0 and is decayed exponentially to a minimum of 0.01 over the first $5,000$ iterations, balancing the initial broad exploration with later-stage exploitation. The replay buffer size is set to $10,000$ to ensure a diverse range of experiences is available for the agent, aiding in the prevention of rapid forgetting and promoting stability in learning updates. The batch size for sampling from the replay memory is configured at 64, the target network update frequency is established at 100 steps.

4.2 Experimental Results

The evaluation of the DRL agent's performance in JSSP instances was conducted focusing on two principal KPIs: makespan and machine utilization. A reduction in makespan directly indicates an enhancement in scheduling efficiency, suggesting that the system can process jobs more rapidly. This increase in throughput also potentially improves customer satisfaction through quicker order fulfillment. Additionally, measuring machine utilization, which indicates the proportion of time machines are actively processing jobs, serves as a critical metric for assessing productivity and minimizing downtime.

Given the widespread utilization of heuristic PDRs for JSSP due to their simplicity and effectiveness under certain scenarios, we conducted a comparative analysis between the performance of the DRL agent and most commonly used PDRs, namely FIFO, SPT, LIFO, LPT, LTWR, MTWR, STPT, LTPT. This comparison is crucial in demonstrating the superior capabilities of the DRL agent over conventional scheduling approaches.

We conducted training and validation experiments on the generated instances 6×6, 10×10, 15×15, 20×20, and 30×20). For each instance size, we randomly generated 100 instances and reported the average objective (makespan) and gap compared to the OR-Tools solutions for our method and various baselines. The gap is calculated using the following formula: $\text{Gap}(\%) = \left(\frac{\text{Objective Value}-\text{Optimal Value}}{\text{Optimal Value}}\right) \times 100$. In these experiments, our DDQN agent was utilized to allocate jobs to machines across different problem instances. The results are tabulated in Table 1, which demonstrates that the DDQN agent consistently outperformed all baseline PDRs across various JSSP instances. It is noteworthy that as the size of the instance increased, the performance of the baseline PDRs diminished significantly, whereas the DDQN agent maintained a stable and relatively superior performance.

Instance		FIFO	SPT	LIFO	LPT	LTWR	MTWR	STPT	LTPT	DDQN
6×6	Obj.	694	656	682	663	676	639	687	651	581
	Gap	42.21%	34.43%	39.75%	35.86%	38.52%	30.94%	40.78%	33.40%	19.06%
10×10	Obj.	1227	1157	1176	1129	1162	1148	1192	1131	1095
	Gap	51.86%	43.19%	45.54%	39.73%	43.81%	42.08%	47.52%	39.98%	35.52%
15×15	Obj.	1892	1803	1833	1756	1823	1795	1877	1746	1577
	Gap	59.26%	51.77%	54.29%	47.81%	53.45%	51.09%	58.00%	46.97%	32.74%
20×20	Obj.	2512	2458	2488	2403	2466	2435	2498	2414	2216
	Gap	61.44%	57.97%	59.90%	54.43%	58.48%	56.49%	60.54%	55.14%	42.42%
30×20	Obj.	3188	3068	3122	3011	3089	3039	3175	3016	2497
	Gap	64.33%	58.14%	60.93%	55.21%	59.23%	56.65%	63.66%	55.46%	28.71%

Table 1: Computational Results.

Figure 2 illustrates the evolution of makespan over time across five distinct JSSP instances with each sub-figure presenting both training and validation performance. While the training lines exhibit an initial exponential decline in makespan, indicating rapid learning, it eventually plateaus; conversely, the validation lines swiftly stabilize, which underscores the model's ability to generalize its learned scheduling strategies effectively across different scales of job shop configurations. Regarding to the machine utilization KPI, we observed at increments of 9% for the 6×6 instance, 12% for the 10×10, 15% for the 15×15, 18% for the 20×20, and reaching up to 21% for the largest 30×20 instance, respectively.

In contrast to heuristic PDRs, which rely on predefined strategies, the DRL agent dynamically adjusts its scheduling solutions in response to the evolving state of the job shop. This entails a potential discovery of more efficient scheduling methodologies not preconceived by heuristic approaches. Consequently, the DRL agent exhibits not only a flexibility in optimizing utilization across varying scenarios but also successfully achieves a balanced optimization of overall system performance.

5 CONCLUSION AND FUTURE WORK

This paper delves into the effectiveness of DRL in optimizing the JSSP through a custom-built DES. Through experimentation and analysis, our findings confirm that the DRL agent possesses the capacity to learn and adapt within various job shop settings, leading to notable improvements in KPIs. Such outcomes highlight DRL's potential as an innovative substitute for traditional PDRs, especially in navigating the complexities presented by dynamic JSSP environments. This investigation enhances the wider comprehension of DRL's utility in combinatorial optimization tasks. Practically, applying DRL in conjunction with DES to address JSSP presents significant implications for sectors that benefit from adaptable and dynamic scheduling approaches.

The experimental results present a strong indication of DRL's capability to refine complex scheduling processes. By capitalizing on DRL's inherent ability to derive insights from interactions within a DES, the

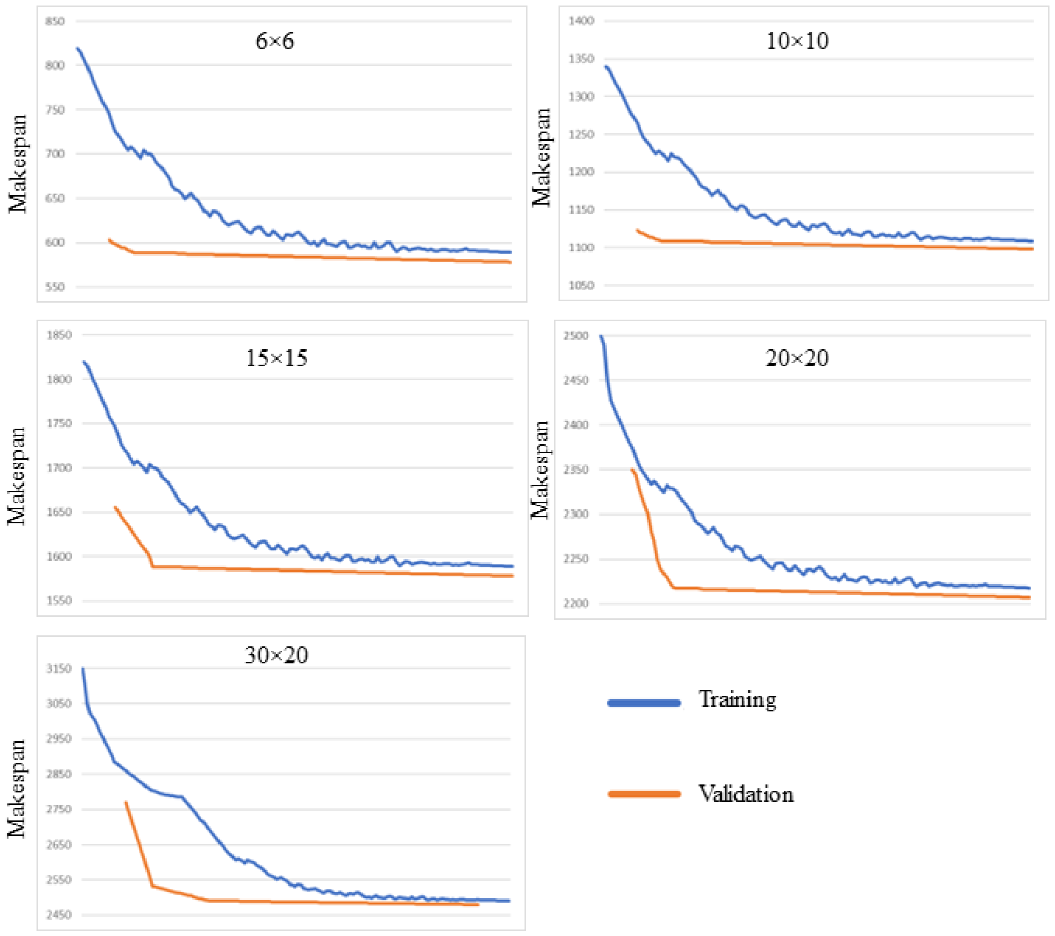

Figure 2: Evolution of Makespan.

DRL agent managed to exhibit enhancements in certain KPIs. Nevertheless, while the advantages seem promising, this approach encounters several hurdles. A notable challenge is the extensive computational resources and time necessary for the DRL agent's training, which escalates with the complexity of the scheduling environment. Additionally, the efficacy of the scheduling solution is substantially influenced by the design of the reward function, state representation, and the DRL model's structure. Another vital issue is ensuring that the agent generalizes well across varied job shop setups without overfitting to particular cases, requiring meticulous adjustment and validation.

In summary, exploring DRL as a solution to the JSSP offers invaluable perspectives on the potential of AI-driven methodologies to redefine conventional scheduling practices. The adaptability, learning capability, and the possibility to discover innovative scheduling strategies underscore DRL as a formidable tool in scheduling optimization. However, overcoming the challenges associated with training intricacies, model design, and ensuring broad generalization is crucial for tapping into DRL's complete applicability in practical scenarios. Future studies could investigate hybrid approaches that merge DRL's strengths with domain-specific heuristics, or delve into advancements in DRL architectures to boost efficiency and scalability.

REFERENCES

Arulkumaran, K., M. P. Deisenroth, M. Brundage, and A. A. Bharath. 2017. "Deep reinforcement learning: A brief survey". *IEEE Signal Processing Magazine* 34(6):26–38.

Belsare, S., E. D. Badilla, and M. Dehghanimohammadabadi. 2022. “Reinforcement Learning with Discrete Event Simulation: The Premise, Reality, and Promise”. In *2022 WinterSim Conference* https://doi.org/10.1109/WSC57314.2022.10015503.

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.

Dantzig, G. B. and P. Wolfe. 1960. “Decomposition principle for linear programs”. *Operations research* 8(1):101–111.

Devanga, A., E. D. Badilla, and M. Dehghanimohammadabadi. 2022. “Applied Reinforcement Learning for Decision Making in Industrial Simulation Environments”. In *2022 WinterSim Conference* https://doi.org/10.1109/WSC57314.2022.10015282.

Gu, S., E. Holly, T. Lillicrap, and S. Levine. 2017. “Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates”. In *2017 IEEE international conference on robotics and automation (ICRA)*, 3389–3396. IEEE.

Hammami, N. E. H., B. Lardeux, A. B. Hadj-Alouane, and M. Jridi. 2022. “Job Shop Scheduling: A Novel DRL approach for continuous schedule-generation facing real-time job arrivals”. *IFAC-PapersOnLine* 55(10):2493–2498 https://doi.org/10.1016/j.ifacol.2022.10.083. 10th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2022.

Han, B.-A. and J.-J. Yang. 2020. “Research on Adaptive Job Shop Scheduling Problems Based on Dueling Double DQN”. *IEEE Access* 8:186474–186495 https://doi.org/10.1109/ACCESS.2020.3029868.

Holthaus, O. and C. Rajendran. 1997. “Efficient dispatching rules for scheduling in a job shop”. *International Journal of Production Economics* 48(1):87–105.

Kiran, B. R., I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani *et al*. 2021. “Deep reinforcement learning for autonomous driving: A survey”. *IEEE Transactions on Intelligent Transportation Systems* 23(6):4909–4926.

Lang, S., F. Behrendt, N. Lanzerath, T. Reggelin and M. Müller. 2020. “Integration of Deep Reinforcement Learning and Discrete-Event Simulation for Real-Time Scheduling of a Flexible Job Shop Production”. In *2020 Winter Simulation Conference (WSC)*, 3057–3068 https://doi.org/10.1109/WSC48552.2020.9383997.

Lang, S., M. Kuetgens, P. Reichardt, and T. Reggelin. 2021. “Modeling Production Scheduling Problems as Reinforcement Learning Environments based on DES and OpenAI Gym”. *IFAC-PapersOnLine* 54(1):793–798 https://doi.org/10.1016/j.ifacol.2021.08.093. 17th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2021.

Manne, A. S. 1960. “On the job-shop scheduling problem”. *Operations research* 8(2):219–223.

Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra *et al*. 2013. “Playing atari with deep reinforcement learning”. *arXiv preprint arXiv:1312.5602*.

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, , , , *et al*. 2015. “Human-level control through deep reinforcement learning”. *nature* 518(7540):529–533.

Panzer, M. and B. Bender. 2022. “Deep reinforcement learning in production systems: a systematic literature review”. *International Journal of Production Research* 60(13):4316–4341.

Sutton, R. S. and A. G. Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., S. Singh, and D. McAllester. 2000. “Comparing policy-gradient algorithms”. *IEEE Transactions on Systems, Man, and Cybernetics*.

Waschneck, B., A. Reichstaller, L. Belzner, T. Altenmüller, T. Bauernhansl, A. Knapp *et al*. 2018. “Deep reinforcement learning for semiconductor production scheduling”. In *2018 29th Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*, 301–306 https://doi.org/10.1109/ASMC.2018.8373191.

Watkins, C. J. and P. Dayan. 1992. “Q-learning”. *Machine learning* 8:279–292.

Xiong, H., S. Shi, D. Ren, and J. Hu. 2022. “A survey of job shop scheduling problem: The types and models”. *Computers & Operations Research* 142:105731.

Zhang, C., W. Song, Z. Cao, J. Zhang, P. S. Tan and X. Chi. 2020. “Learning to dispatch for job shop scheduling via deep reinforcement learning”. *Advances in neural information processing systems* 33:1621–1632.

Zhang, W. and T. G. Dietterich. 1995. “A reinforcement learning approach to job-shop scheduling”. In *Ijcai*, Volume 95, 1114–1120.

AUTHOR BIOGRAPHIES

BULENT SOYKAN is a Postdoctoral Researcher at the School of Modeling, Simulation, and Training, University of Central Florida. He received his Ph.D. from Old Dominion University. His research interests include combinatorial optimization, digital twins, reinforcement learning. His email address is bulent.soykan@ucf.edu.

GHAITH RABADI is a Professor and Graduate Programs Director at the School of Modeling, Simulation, and Training, University of Central Florida. He received his Ph.D. in Industrial Engineering from University of Central Florida. His research interests include complex systems modeling, logistics, supply chains, optimization, simulation and AI to find optimal and near-optimal solutions. His email address is ghaith.rabadi@ucf.edu.