Proceedings of the 2022 Winter Simulation Conference
B. Feng, G. Pedrielli, Y. Peng, S. Shashaani, E. Song, C.G. Corlu, L.H. Lee, E.P. Chew, T. Roeder, and P. Lendermann, eds.

APPLIED REINFORCEMENT LEARNING FOR DECISION MAKING IN INDUSTRIAL SIMULATION ENVIRONMENTS

Ashwin Devanga
Emily Diaz Badilla
Mohammad Dehghanimohammadabadi

Department of Mechanical and Industrial Engineering
Northeastern University
360 Huntington Avenue
Boston, MA 02115, USA

ABSTRACT

The industrial sector is going through a digital transformation with a high emphasis on Artificial Intelligence-driven operations. In this transformation, the combination between simulation and machine learning has been a key enhancer to provide state-of-the-art solutions to complex systems by training algorithms over virtual representations of industry. In recent years, Reinforcement Learning (RL) has gained traction in this field and shown success in solving sequential decision-making problems. The purpose of this paper is to show how to implement simulation-based Reinforcement Learning. The proposed framework integrates Simio, as a discrete-event simulation environment, and Python, to include the RL algorithm. To demonstrate the applicability of this framework, a job-shop scheduling problem under different scenarios is tested and its results are compared with benchmark heuristic dispatching rules.

1 INTRODUCTION

Today, one of the prevalent technologies is Digital Twin (DT), which is a replica of a real-world system. The DT creates an identical, and virtual version of a product, service, or process informed by real-life data. It enables enterprises to simulate and configure their system without disrupting the real-world environment. Due to its promises and the agility it offers, DT is positioned as one of the top 5 strategic technology trends for 2021 and 65 percent of executives expect their organization's investment in the intelligent DT to increase over the next three years (Accenture 2020).

Simulation software packages are widely accepted tools to implement DT. Although these packages can capture the dynamics of business processes in detail, they fail to utilize Artificial Intelligence (AI) to its' full potential and process the data to reveal actionable insights. In addition, applying AI to the simulation packages requires different skill sets with expert knowledge which is challenging for many enterprises.

Therefore, it is essential to equip the existing DT platforms with advanced analytical solutions empowered by AI to make systems autonomous. The two technologies effectively capture systems' complexities and leverage data analytics to support enterprises to gain operational excellence.

This can be achieved by developing a practical and user-friendly implementation of AI with the simulation environment to support DT. Such a platform requires to seamlessly connect a simulation environment with an AI model to i) train and test the best AI practices for their systems, ii) transform large quantities of data into meaningful insights, and iii) recommend the best plan for future challenges.

An area of AI that has been gaining traction on DT related problems is Reinforcement Learning (RL). This machine learning family of algorithms consist of an agent learning to take the best actions to achieve

a goal in an uncertain, potentially complex environment. This is a technique that particularly works well with simulation as it requires large number of trials-and-errors to train.

This paper provides a showcase on how to integrate an RL agent with a Discrete Event Simulation (DES) environment. This integration uses Simio software package with a Python coded agent that learns to prioritize jobs in a job-shop problem. To complete this, an API is developed to seamlessly transfer data between Python and Simio in both directions. In addition, multiple test scenarios are conducted to analyse different problem sizes, and two different RL agents Q-Learning and Double Deep Q-Learning Networks(DDQN) are showcased. The experimental results indicate how an RL agent can learn the dynamics of the model and obtain a policy to optimally make decisions.

The rest of the paper is organized as follows: Section 2 provides an overview of applications and evolution of RL in conjunction with simulation; Section 3 provides the framework for the environment and agents algorithms; Section 4 shows the results of the experiments and compares their performance to benchmark heuristic dispatching rules. This paper is concluded in section 5 by providing a summary of work done and possibilities in the future.

2 LITERATURE REVIEW

There is a strong body of literature that discusses a variety of simulation optimization approaches. In recent years, the increasing popularity of machine learning (ML) methods triggered many developments and have opened a new avenue of research in the filed. As a key methodology of AI, ML is a preferred approach to solving large-scale data analysis. Therefore, ML is a perfect fit for simulation environments where an abundance of high-quality data is generated. Since this paper emphasizes using RL, a brief review of existing simulation papers that apply this ML method is provided below.

2.1 An overview of Reinforcement Learning

RL is a concept evolved from work done by Rich Sutton and Andy Barto in the 1980's (Powell 2019). However, its foundation comes from Markov Decision Process (MDPs) dated back in the 1950s. The inspiration for its development came from solving the problem of a mouse exploring a maze (Powell 2019). Furthermore, this field attracted significant attention when in 2016 it was credited with solving the Chinese game of Go using AlphaGo, the first system to beat world-class players (Powell 2019).

The main idea behind RL is to capture the most important aspects of the problem facing an agent interacting with its environment to achieve a goal (maximize a reward) (S. Sutton and G. Barto 2014). The agent's goal is to push the boundaries of what is currently possible through learning the optimal mapping of situations to actions through trial-and-error (Nian et al. 2020). The two major components of an RL model are (i) *agent* and (ii) *environment* that interact using actions, observations, and rewards. By interacting with the environment, an agent can take an action and receive the next observed states and its corresponding reward (2.2). After trying a variety of actions, the agent can progressively favor those that appear to be best (S. Sutton and G. Barto 2014) and learn the optimal *policy*.

In recent years, there has been an increasing interest in RL driven by its successful applications. One of the main developments was in 2013, when Mnih et al. introduced the deep Q-learning (DQN) framework (Mnih et al. 2013). They introduced a combination of Deep Learning and RL (DRL) to play seven Atari games from raw image pixels. Their model was able to achieve human player-like performance and even surpass them in three of the games tested.

In its early years, there were few RL use-cases in industrial or commercial settings compared to the ones of video games (Gros et al. 2020). Due to its success, RL is now becoming one of the promising Machine Learning approaches to address real-life problems in multiple industries such as healthcare (Yu et al. 2019), manufacturing (Nian et al. 2020), supply chain (Barat et al. 2019), among many.

In the healthcare domain, RL is being used to develop effective treatment regimes that can dynamically adapt to the varying clinical states and improve the long-term benefits of patients, called Dynamic Treatment

Regimes. RL is capable of achieving time-dependent decisions on the best treatment for each patient at each decision time, thus accounting for heterogeneity across patients (Yu et al. 2019).

RL has also demonstrated success in the process control area. In 2018, a cooling system for a Data Center control policy was modeled using DRL framework, leading to an 11% cooling cost saving on the simulation platform compared with a manually configured baseline control algorithm (Li et al. 2020).

Another application that has shown promise is supply chain management. In 2019, Barat et al. proposed a framework with an RL controller integrated with an actor-based simulation of a grocery retailer networked system, in order to enable deployment of the RL agent in the real system with minimal live operation experimentation (Barat et al. 2019). The objective of the model was to optimally regulate the availability of the entire product range in each store at all times, subject to the Spatio-temporal constraints imposed by available stocks, labor capacity, truck capacity, transportation times, and available shelf space for each product in each store. They concluded that this option is computationally feasible and effective compared to traditional approaches (Barat et al. 2019).

2.2 Importance of Simulation for RL

To effectively train an RL model, the agent needs to receive an abundance of trials and errors from the environment. To overcome this problem, simulation becomes a cost-effective and risk-free alternative that can be directly transferable to the real process (Figure 2.2). As an example, DRL with simulated environments has been used for room-navigating robot policy training with successful deployment to the real robot (Tai et al. 2017). It helps in the process of first training the RL agent in a *process simulator* to obtain general knowledge of the process (Nian et al. 2020). DRL has already demonstrated to perform well in real-time decision making in car manufacturing process (Gros et al. 2020). This work combined a simulation model with DRL algorithms and found this model to be performing extremely well and the resulting strategies provide near-optimal decisions in real-time, while alternative approaches were either slow or with poor quality strategies (Gros et al. 2020).

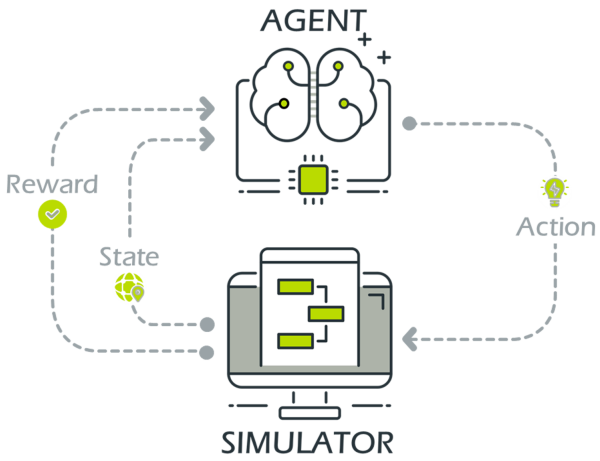


Figure 1: Reinforcement Learning with a simulated process framework.

Simpy is an open-source package for process-based DES available in Python available since 2002 and widely used for simulation modeling. In the Supply Chain industry, it has been used for complex tasks such as routing and schedule simulation (Pinho et al. 2020). In the telecommunications industry, Simpy has been leveraged to build an event-driven simulator to perform small and large scale simulations on architectures such as Cloud-Fog Radio Access Network (CF-RAN) used to decide the best architecture for energy-efficient access networks to mobile users in the upcoming 5G technology (Tinini et al. 2020).

Another example of an open-source Python-based library is OR-Gym (Hubbs et al. 2020), which provides RL environments consisting of classic operations research and optimization problems. From the experiments by Gros et al., RL approach outperforms the benchmark in many of the more complex environments where uncertainty plays a significant role. Such platforms would ensure accessibility of RL to solve industrial optimization problems where models do not exist or are too expensive to compute online (Gros et al. 2020).

In recent years, commercial software packages have been bridging simulation features with RL. Pathmind is a SaaS platform that provides RL modeling with simulation leveraging AnyLogic simulation software. One use case provided is a coffee shop simulation built to train a barista to make correct operational decisions and improve efficiency that directly affects customer service time (Farhan et al. 2020). There have also been advancements in making RL available without the need for coding. This is because of the fact that simulation model developers may lack expertise in the software tools and coding abilities needed for the development of Machine Learning algorithms from scratch (Greasley 2020). The authors developed a DES that incorporates the use of an RL algorithm to determine an approximate best route for robots in a factory moving from one physical location to another whilst avoiding collisions with fixed barriers. The study showed how the object modeling and graphical facilities of the Simio commercial software package enable an RL capability without the need to use program code or require an interface with external RL software (Greasley 2020).

Although many studies such as (Peyman et al. 2021) and (Kabadayi and Dehghanimohammadabadi 2022) tried to integrate DES models with optimizers to perform simulation-optimization, there is a lack of successful RL implementation. This paper also contributes to the literature by demonstrating how to connect an open-source coding language such as Python, to a DES model. This work will create a new avenue of research to apply multiple RL agent settings and libraries. In addition, the DES package can model any system to its details and enable the RL agent to provide accurate actionable insights.

3 REINFORCEMENT LEARNING FRAMEWORK

The proposed simulation-based RL model in this study is an integration of Simio, a commercial off-the-shelf (COTS) software package, and Python, an open-source programming language. To implement this connection, an API is coded in C# to enable the RL agent in Python easily interact with its simulated environment built in Simio. Using this connection, the RL agent can perform actions in steps and receive the corresponding rewards from the simulation model. The interaction continues in a recursive manner until the RL agent gradually completes its training and converges to an optimal policy.

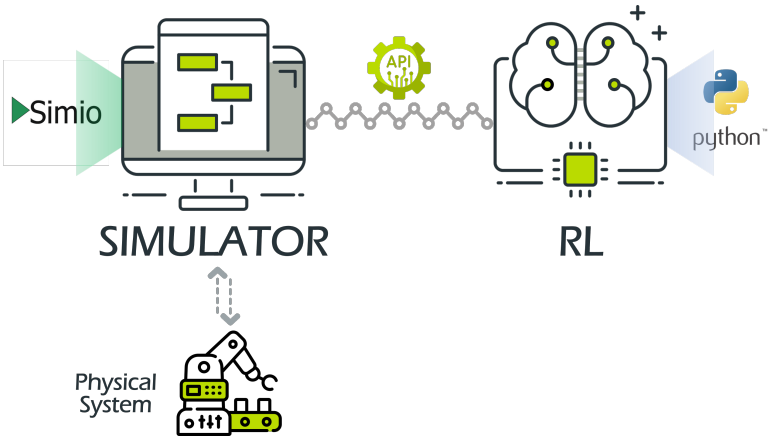


Figure 2: Simio-Python Integration.

It needs to be noted that the study represents a proof-of-concept to demonstrate how to implement a simulation-based RL in Simio. Therefore, the developed simulation examples discussed below are simplified to enable readers to focus on this integration feasibility and get practical insights. In addition, the developed framework is independent of the simulation complexity and the RL agent algorithm. Depending on the users' needs, the simulation can include any kind of architecture to represent a process in real-life. In a similar way, the applied RL algorithm can be adjusted in Python based on tabular or function approximation methods.

3.1 Simulation Environment

The simulation model is a job scheduling problem that includes a single machine with multiple jobs with different characteristics. Figure 3 shows the simulated environment built in Simio. There are 30 jobs to be processed each with its unique arrival time, processing time, and setup time (see Table 1). The goal is to train an RL agent that can learn how to prioritize the jobs based on their characteristics (features) and optimize the long-term reward. In this case, the objective is set to minimize the average time in the system after processing all jobs.

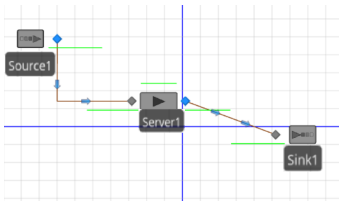


Figure 3: Simulation model in Simio.

Table 1: Simulation model distributions.

Characteristics	Distributions/Pattern
Arrival	Arrival Table
Processing Time	$Normal(\mu, \sigma)$
Setup Time	$Normal(\mu, \sigma)$
Due Date	Deterministic

3.2 RL Agents

The RL agents are the algorithms which train on the simulation and learn to optimise the required parameters. It has multiple implementations. The ones used here are Q-Learning and DDQN.

3.2.1 Q-learning

Q-learning (QL) is one of the most popular RL algorithms. It is a model-free algorithm that learns the value of an action in a particular state. It provides agents with the capability of learning to act optimally in Markovian domains by experiencing the consequences of actions, without requiring them to build maps of the domains (Watkins and Dayan 1992). Q refers to the function of RL that calculates the expected rewards for an action taken in a given state. Each Q-table value will be the maximum expected future reward that the agent will get if it takes that action at that state. By trying all actions in all states repeatedly, it learns which are best overall, judged by long-term discounted reward (Watkins and Dayan 1992). The steps of the algorithm are shown in Algorithm 1.

Algorithm 1 : Q-learning (off-policy TD control) for estimating Policy Values.

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathscr{S}^+, a \in \mathscr{A}(s)$, arbitrarily except that $Q(\text{terminal},\cdot) = 0$

for episode **do**
  Initialize S
  for step of episode **do**
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
    $S \leftarrow S'$
  end for
end for

3.2.2 Double Deep Q Networks (DDQN)

DDQN is an algorithm that leverages deep learning by using two neural networks to approximate Q-learning function. Q-learning algorithm is known to overestimate action values under certain conditions. The idea of Double Q-learning is to reduce over-estimations by decomposing the max operation in the target into action selection and action evaluation. Although not fully decoupled, the target network in the Deep Q Network (DQN) architecture provides a natural candidate for the second value function, without having to introduce additional networks (Hasselt et al. 2016). The steps of the process are shown in algorithm 2.

Algorithm 2 : Double Deep Q-learning Network (DDQN).

Initialize Q^A, Q^B, s
repeat
  Choose a, *based on* $Q^A(s,\cdot)$ *and* $Q^B(s,\cdot)$, *observer*, s
  Choose (e.g. random) either UPDATE(A) or UPDATE(B)
  if $UPDATE(A)$ **then**
    $Define\ a^* = argmax_a Q^A(s',a)$
    $Q^A(s,a) \leftarrow Q^A(s,a) + \lambda(s,a)(r + \gamma Q^B(s',a^*) - Q^A(s,a))$
  else if UPDATE(B) **then**
    $Define\ b^* = argmax_a Q^B(s',a)$
    $Q^B(s,a) \leftarrow Q^B(s,a) + \lambda(s,a)(r + \gamma Q^A(s',b^*) - Q^B(s,a))$
  end if
  $s \leftarrow s'$
until End

3.3 Simio Python API

To transfer data between Python and Simio, an API was developed so that it can communicate between the simulation environment and the RL agent. There are two versions of this API. One which can call a Python function (in this case, RL algorithm) from Simio (CallPython API, a user-defined step), and the other calls Simio from Python (CallSimio API). Both these APIs have advantages and disadvantages when it comes to using them for RL.

All results showcased in this paper use CallPython API which is embedded in the model as a user-defined step in Simio and set inside the processes of the model. This is what calls the RL agent written in Python to choose the priority value of that job. The experiment is set to run the model for the expected number of iterations of training. The APIs are written in C# as Simio itself is a standalone C# application.

4 EXPERIMENTATION AND RESULTS

To evaluate the performance of the developed model, multiple scenarios are conducted to understand how the RL agent performs under different conditions. The following section elaborates on experiments and the obtained results.

4.1 Base model

The base model for this work consists of 4 possible actions, or priorities in this context, to be taken by the agent. This can be an integer value between 1 to 4 that represents the priority in the queue. Therefore, jobs get processed by the server based on the priority they received by the RL agent. The reward function is focused on *minimizing* the average time in the system (cycle time) which was translated to a *maximization* for the RL agent as shown in equation (1).

$$R(i) = CT_{max} - CT(i), \quad \forall i \in I \tag{1}$$

where CT_{max} is an estimated worst cycle time that the model can take, and $CT(i)$ represents the achieved cycle time by the end of episode i. In every episode, agent updates its policy, and the simulation model is executed to observe its corresponding reward (R_i). On the base model, the reward is determined at the end of each episode and no intermediate reward is considered between action steps $(R_i nt = 0)$. I represents number of all Jobs.

Since both Q-Learning and DDQN are off-policy Learning Algorithms, they benefit from exploration. This means that they learn to estimate good decisions by looking at bad decisions as well. Therefore we implemented the epsilon greedy algorithm.

Figure 4 shows how epsilon decay was defined for both models over the 800 episodes run. For the first 200 episodes, the agent is set to fully explore the environment. After that, the exploration and exploitation decision process for Q-learning and DDQN differs in the sense that Q-learning's epsilon is set to decay after that until reaching zero and on the other hand, DDQN would not reach zero and around episode 450 it stays at a value of approximately 0.1.

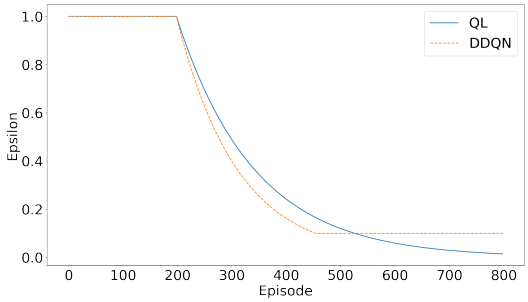
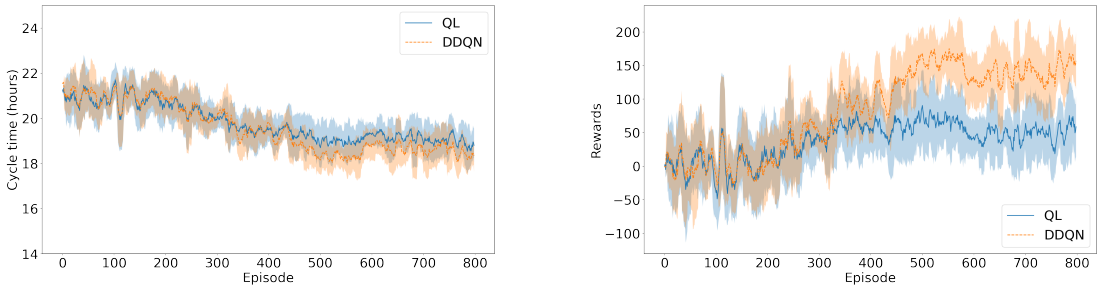


Figure 4: Epsilon decay for base model with 4 actions.

The results are shown in Figure 5 where algorithms are performing similarly and converging on a final value of around 18 hours for cycle time. A slight difference between the models is due to the fact that Q-learning took the average over the whole list of cycle times, but DDQN used the average cycle time from the first 200 episodes only.

4.2 Problem complexity

In this section, the action space of the problem is modified for two new experiments where the number of discrete actions are increased. The objective is to demonstrate how the problem can be scale to a wider

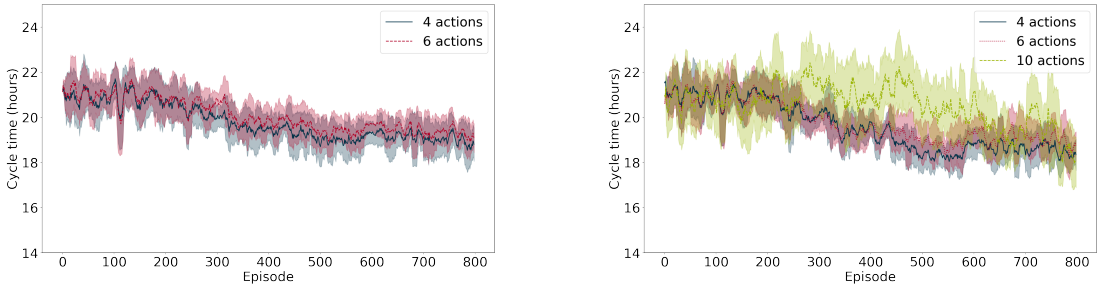


(a) Cycle times (b) Rewards

Figure 5: Training results of base model with 4 actions.

range of actions to be taken. For this, two new experiments are introduced: Q-learning with 4 and 6 actions and DDQN with 4, 6, and 10 actions. Q-learning was not explored for the largest set of actions as it becomes too computationally expensive.

Figure 6 shows Q-learning 4 and 6 actions are performing similar. In the case of DDQN, when increasing to 10 actions, the model becomes less stable and is not converging over the 800 episodes.



(a) Q-learning (b) DDQN

Figure 6: Cycle time for training data by number of actions.

4.3 Reward function modification

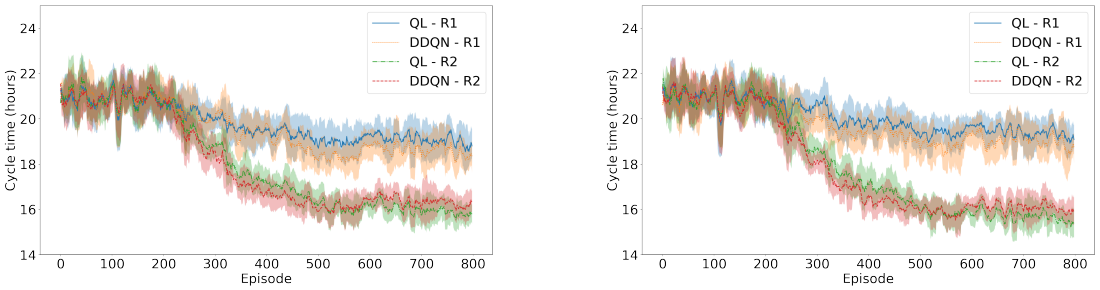
To provide more information to the agent, a modified version of the previous reward function was implemented for both Q-learning and DDQN as well as tested with different number of actions. Initially, the reward was set to 0 for each step when the job was assigned a priority value. The final reward was explained above in equation 1. Instead we pressurise the agent to learn quickly by providing a feedback in every step. The implementation is shown in equation (2).

$$R_{int}(i,j) = -[(PT_{max}/n_a) \times pr(j) - PT(j)]^2, \quad \forall i \in I \tag{2}$$

where $R_{int}(i,j)$ represents the intermediate reward for step j on episode i, PT_{max} indicates the estimated maximum processing time for all jobs, n_a is for number of possible actions, $pr(j)$ represents the priority of job j (step j), and $PT(j)$ corresponds to the processing time of job j.

Having feedback between each step helps the agent learn quicker. It also gets a large amount of data feeded into it which can be used to implement multi-objective optimisation too.

The results of this change can be seen in Figure 7, where initial reward function (R1) and new reward function (R2) are compared. In all experiments, the reward function with intermediate knowledge is always far more superior than the initial reward. It also shows that for our largest set of actions, it can learn and return the best results for cycle time with a final value of approximately 15 hours.



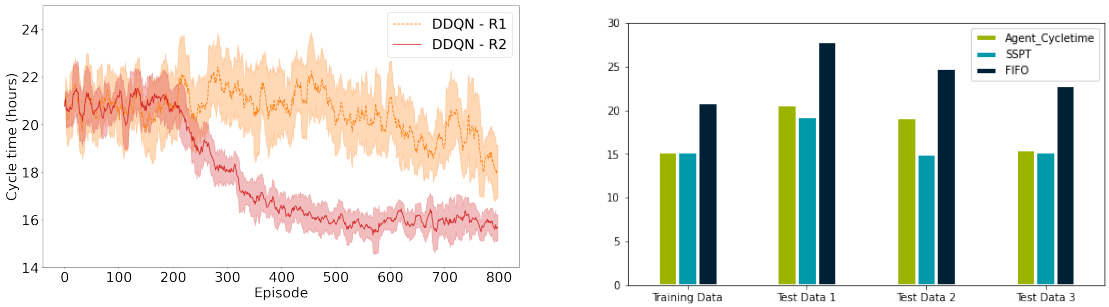
(a) 4 actions (b) 6 actions

Figure 7: Cycle time for training data with different reward functions.

4.4 Evaluation

In this section, the results obtained are compared to First-In-First-Out (FIFO) and Shortest Setup-and-Processing-Time (SSPT) dispatching rules. SSPT was considered as it has shown to be a good performing heuristic method for job-shop scheduling (Dehghanimohammadabadi 2016). The best performing RL agent is a DDQN trained with 10 actions (10 possible priority values). This agent is tested on 3 different test datasets and the results are shown in Figure 8.

The agent outperforms FIFO by a great margin. Its performance can be compared to SSPT in most cases. If trained with more precisely tuned parameters and longer episodes, the agent can outperform SSPT as well. Evidence for this is seen in the training dataset where it already slightly outperforms SSPT.



(a) Training Results for 10 Actions (b) Evaluation of trained DDQN agent

Figure 8: Evaluation of Trained agent over test datasets

5 CONCLUSIONS

In certain scenarios, simulation models need to be empowered with external environments to support advanced calculation, optimization, or evaluation (Dehghanimohammadabadi and Keyser 2017). This highlights the importance of *intelligent simulation* modeling, where a DES platform is connected with a powerful programming language such as R, Python, or MATLAB. Due to the ongoing growth of machine

learning techniques and optimization algorithms, this combination becomes even more necessary than ever. However, there is a lack of tutorials on how to efficiently combine simulation software with data science programming languages such as Python.

Python is widely used to solve statistical and scientific problems. Combining simulation software with Python will provide enormous advantages for experimentation, optimization, results analysis, and professional data visualization. Additionally, it is an open-source language supplemented with multiple libraries for scientific computing, machine learning, optimization, data science, and big data.

The objective of this paper was to showcase the possibilities of combining RL with DDES and using it for self-healing Digital Twin models. In a series of experiments it is shown that a trained agent combats most situations and also continues to learn. This means that the decisions made by the agent will change over time as it evolves further. It also has the advantage of looking at scenarios objectively and so it can identify ways to optimize the operation in ways unknown.

This work can be extended in many ways:

- **Simulation Model complexity:** include simulation model with complex operation with an RL agent facilitate actions on the models. The area of applications could be in a supply chain where the agent determines the replenishment policy of the network, a warehouse model where robots are routed by RL, or a manufacturing setting where all servers are controlled by a centralized RL agent.
- **Advanced RL agents:** This paper can be further advanced by considering multiple RL algorithms especially those with capabilities to deal with continuous action space. The current work considers a limited number of actions, 4, 6, and 10 whereas, in reality, the number of jobs and actions is enormous. Therefore, combining simulation with advanced RL algorithms such as Soft-Actor Critique (SAC) models allows the model to solve problems with continuous action-space.
- **Data-driven model:** A successful implementation of Digital Twin requires a data-driven simulation model that is connected to a stream of data from data sources such as ERP systems. From a practical point of view, combining such a data-driven simulation with RL algorithms can help the system to get actionable insights in real-time.

REFERENCES

Accenture 2020. "Technology vision 2021 leaders wanted". Accessed 22nd March 2022.

Barat, S., H. Khadilkar, H. Meisheri, V. Kulkarni, V. Baniwal, P. Kumar, and M. Gajrani. 2019. "Actor Based Simulation for Closed Loop Control of Supply Chain Using Reinforcement Learning". In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. May 13th-17th, Montreal, Canada, 1802–1804.

Dehghanimohammadabadi, M. 2016. *Iterative Optimization-based Simulation (IOS) with Predictable and Unpredictable Trigger Events in Simulated Time*. Ph. D. thesis, Western New England University.

Dehghanimohammadabadi, M., and T. K. Keyser. 2017. "Intelligent simulation: Integration of SIMIO and MATLAB to deploy decision support systems to simulation environment". *Simulation Modelling Practice and Theory* 71:45–60.

Farhan, M., B. Göhre, and E. Junprung. 2020. "Reinforcement Learning in Anylogic Simulation Models: A Guiding Example Using Pathmind". In *Proceedings of the 2020 Winter Simulation Conference*, edited by K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, 3212–3223. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Greasley, A. 2020. "Implementing Reinforcement Learning in Simio Discrete-Event Simulation Software". In *Proceedings of the 2020 Summer Simulation Conference*. July 20th-22nd, Virtual, Spain, 1-11.

Gros, T., J. Groß, and V. Wolf. 2020. "Real-Time Decision Making for a Car Manufacturing Process Using Deep Reinforcement Learning". In *Proceedings of the 2020 Winter Simulation Conference*, edited by K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, 3032–3044. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Hasselt, H. v., A. Guez, and D. Silver. 2016. "Deep Reinforcement Learning with Double Q-Learning". In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*. February 12th-17th, Phoenix, Arizona, 2094–2100.

Hubbs, C. D., H. D. Perez, O. Sarwar, N. V. Sahinidis, I. E. Grossmann, and J. M. Wassick. 2020. "OR-Gym: A Reinforcement Learning Library for Operations Research Problem". *CoRR* abs/2008.06319:0.

Kabadayi, N., and M. Dehghanimohammadabadi. 2022. "Multi-objective supplier selection process: a simulation–optimization framework integrated with MCDM". *Annals of Operations Research* 1(1):1–23.

Li, Y., Y. Wen, D. Tao, and K. Guan. 2020. "Transforming Cooling Optimization for Green Data Center via Deep Reinforcement Learning". *IEEE Transactions on Cybernetics* 50(5):2002–2013.

Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. 2013. "Playing Atari with Deep Reinforcement Learning". *CoRR* abs/1312.5602:1–9.

Nian, R., J. Liu, and B. Huang. 2020. "A review On reinforcement learning: Introduction and applications in industrial process control". *Computers & Chemical Engineering* 139:106886.

Peyman, M., P. Copado, J. Panadero, A. A. Juan, and M. Dehghanimohammadabadi. 2021. "A tutorial on how to connect python with different simulation software to develop rich simheuristics". In *Proceedings of the 2021 Winter Simulation Conference*, edited by S. Kim, B. Feng, K. Smith, S. Masoud, Z. Zheng, C. Szabo, and M. Loper, 1–12. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Pinho, T., J. Coelho, P. Oliveira, B. Oliveira, A. Marques, J. Rasinmäki, A. Moreira, G. Veiga, and J. Boaventura-Cunha. 2020. "Routing and schedule simulation of a biomass energy supply chain through SimPy simulation package". *Applied Computing and Informatics* 17(1):36–52.

Powell, W. B. 2019. "From Reinforcement Learning to Optimal Control: A unified framework for sequential decisions". *CoRR* abs/1912.03513:1–50.

S. Sutton, R., and A. G. Barto. 2014. *Reinforcement Learning: An Introduction Second Edition: 2014, 2015*. A Bradford Book.

Tai, L., G. Paolo, and M. Liu. 2017. "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation". *IEEE/RSJ International Conference on Intelligent Robots and Systems* 1:31–36.

Tinini, R. I., M. R. P. dos Santos, G. B. Figueiredo, and D. M. Batista. 2020. "5GPy: A SimPy-based simulator for performance evaluations in 5G hybrid Cloud-Fog RAN architectures". *Simulation Modelling Practice and Theory* 101:102030.

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

Yu, C., J. Liu, and S. Nemati. 2019. "Reinforcement Learning in Healthcare: A Survey". *CoRR* abs/1908.08796:1–32.

AUTHOR BIOGRAPHIES

ASHWIN DEVANGA is a graduate student pursuing MS in Data Analytics Engineering at Northeastern University, Boston, MA, USA. He has extensive research experience in the field of theoretical RL and is now working on implementing these ideas in industrial engineering and operations research problems. His email is devanga.a@northeastern.edu.

EMILY DIAZ BADILLA is a graduate student pursuing MS in Data Analytics Engineering at Northeastern University, Boston, MA, USA. She has 5 years of working experience as Senior Data Scientist in Management Consulting industry. Her current research includes applied Reinforcement and Deep Learning. Her email address is diazbadilla.e@northeastern.edu.

MOHAMMAD DEHGHANIMOHAMMADABADI is Associate Teaching Professor of Mechanical and Industrial Engineering, Northeastern University, Boston, MA, USA. His research is mainly focused on developing and generalizing simulation and optimization frameworks in different disciplines. This article is part of his initiatives to develop a framework to integrate Discrete Event Simulation with RL. His e-mail address is m.dehghani@northeastern.edu.