Parameter-Free Approximation Method for Controlling Discrete Event Simulation by Reinforcement Learning

Daniel Pasterk*, Andreas Körner

Institute of Analysis and Scientific Computing, TU Wien, Wiedner Hauptstraße 8-10, 1040 Vienna, Austria
* daniel.pasterk@tuwien.ac.at

SNE 33(1), 2023, 31-34, DOI: 10.11128/sne.33.sn.10635
Selected ASIM SST 2022 Postconf. Publication:2023-02-01;
Received Revised Improved:2023-02-27; Accepted:2023-03-10
ISSN Print 2305-9974, Online 2306-0271, www.sne-journal.org

Abstract. A novel kNN-based approximation method for Reinforcement Learning (RL) is used to control and optimize a Discrete Event Simulation (DES). The method does not require design parameters, is suitable for unknown and new simulation environments, and can handle irregular and partially sparse state space. We show in a demonstrative queuing simulation that the method is more robust than artificial neural networks and achieves comparable performance.

Introduction

In this contribution we want to show how Discrete Event Simulations can be controlled and optimized using reinforcement learning. From a general perspective, it is promising to optimize simulations using RL. It provides the ideal framework to represent time-dependent decision problems in a very natural approach, while the simulations can generate sufficient d ata. Changes in the dynamics of environment can also be addressed via different exploration strategies. Although a large number of powerful algorithms have emerged since the 1960s, the broad availability of powerful computational resources, has extended its applicability in many areas. Starting with the impressive achievements of DeepQ-Networks [8] in 2013, this form of machine learning was becoming increasingly popular. Controlling a simulation with RL is follows the framework of Markov Decision Processes (MDPs), where an agent interacts with an environment. The agent receives the state of the environment and decides on an action, while the environment responds with a new state and a reward. The agent tries to choose his actions in a way that the cumulative reward is maximized. The simulation therefore represents the environment, in which the agent interacts within the described loop.

While the basic architecture is very clear, the difficulty often arises at dealing with the formal requirements of the framework. It is a important requirement to represent the simulation as an MDP. While the optimization goals and control options can often be represented very directly with a corresponding reward and action set, the so-called Markov property must be fulfilled for the state variable: The entire (relevant) information must be able to be encoded in a single state which has to be independent of its history. In order to fulfill these preconditions feasibly, there are various approaches: The simulation can already be developed with the background of this requirement and the corresponding state variables can be equipped with a well-suited encoding. Though it has to be mentioned here that special knowledge about the nature of the simulation is relevant. This somewhat undermines the claim of a model-free method. As an alternative to the "handcrafted" states, one can take the internal simulation variables directly. These can basically be used as states in the MDP framework and should satisfy the Markov property, but it sadly leads to a very high complexity. In addition, the resulting state space is very irregular - many combinations of the individual components of the state vector are not even possible due to the simulation logic. Large areas of the state space are therefore not used, while in other areas there is a lot of information in a very dense form. Hence, a suitable approximation method has to be found, which can handle these specific properties sufficiently well. This contribution presents a non-parametric approximation method based on the idea of k-nearest-neighbor classification algorithm, which is very well suited for the mentioned challenges. We demonstrate the method in the context of classical Q-learning and Monte-Carlo algorithms.

1 Related Work

We want to give a small selection of publications, which illustrate the successful application and then go into specific l imitations: I n 1 998, Mahadevan and Theocharous [7] use the customized algorithm "SMART" to optimize transfer lines from a fab including predictive maintance. The results show superior performance compared to the classical 'KANBAN' heuristic. In Waschneck et al. (2018) [11], DQN is successfully applied to control a digital twin of a semiconductor fab. The DQN actions control specific dispatch heuristics to increase overall performance. Shuhui Qu, Jie Wang, and Shivani (2016) [9] use RL to solve a dispatching problem in factories. A simulation was created special as MDP and the learned rules are more cost effective than most well-known heuristics. Doltsinis, Ferreira, and Lohse (2014) [3] use RL to optimize a production startup in an 'automatic assembly station'. The use of a batched Q-learning algorithm shows the general suitability of this decision-supported approach. Some selection of general approaches, which focuses on the general connectivity of RL with simulations: Capocchi et al. (2022) [5] show how DEVS properties can be used within a RL integration focusing on temporal, hierarchical and multi-agent aspects. Lang et al. (2021) [6] demonstrate the application of DEVS to a production planning problem. In this approach, it is shown how control can be performed using a standardised interface (OpenAi Gym). In Greasley (2020) [4], various software products from simulation and machine learning are examined for connectivity. Some interesting considerations are also made about the structure of a useful connectivity component. In Choo et al. (2020)[2], the general problem that discrete event simulations cannot be directly translated into MDP is pointed out. It is proposed to translate the state space and action space of the simulation into their counterparts in the RL formalism using equivalence classes. The approach sounds interesting, but unfortunately no statement is made about the concrete form of the relationship.

2 Method

2.1 Architecture

The architecture of our setup can be summarized in Figure 1. For the interface of the simulation we use the interface of OpenAi-Gym [1], by which the states, actions and the reward are exchanged. The simulation-internal variables are made accessible to the agent as a state vector without further processing. The agent has no further information about the model apart from the state vector and the simulation is technically controlled by this model-free agent. Both the implementation of the code for the approximation and learning feature, as well as the implementation of the simulation was done in the programming language Julia. The framework "SimJulia" was used for the simulation and the modeling follows the agent-based paradigm.

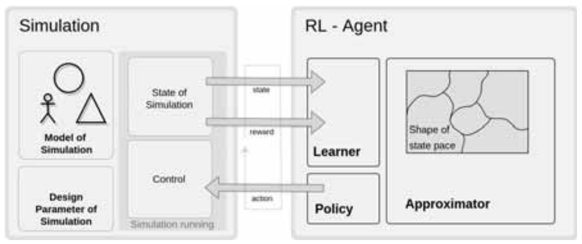

Figure 1: General overview of the architecture of the simulation and the RL agent. The approximator must represent the underlying state space sufficiently well.

2.2 Approximation Method for Reinforcement Learning

On the agent side, we strongly separate between the elements of "learner", "approximator" and "policy". This is also illustrated in Figure1. For the case-study below, we use the update rules from classical Q-learning. For the state space S, action space A, $\alpha \in \mathbb{R}$, $\gamma \in \mathbb{R}$ and a possible reward $r \in \mathbb{R}$ the update rule can be written as

$$q(s,a) := q(s,a) + \alpha(r + \gamma \max_{a \in A(s')}(q(s',a)) - q(s,a)). \tag{1}$$

with $s \in S$ and $a \in A$. We use this update rule in connection with various approximation methods such as "State Aggregation", "Tile-Coding" and "Linear Regression".

We also use Monte-Carlo-1st-Visit [10] and a Deep Q-Network [8] with two different capacities as a reference.

Since A is finite, an optimal policy π is immediately calculated as

$$\pi(s) := \text{argmax}_{a \in A(s)} q(s,a), \tag{2}$$

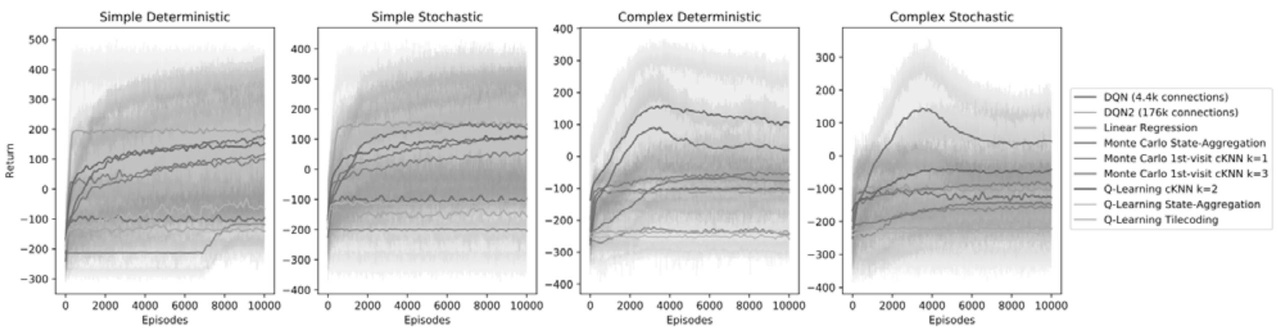

Figure 2: Performance of k-NN approximation in combination with Q-Learning and first-visit Monte Carlo compared to other common approximation methods, in simple-deterministic ($n = 20$), simple stochastic ($n = 20$), complex-deterministic ($n = 20$), and complex-stochastic tasks ($n = 20$).

where ties are broken randomly, after having computed the action-value function q.

Our method for approximating $q(s,a)$ is based on the idea that the collected interactions between agents and environment are grouped into (k-d)-Trees to the respective action $a \in A$. The RL update rules provide an error that is added to the current estimate of an existing query. To obtain $q(s,a)$ in a query, every KD-tree is searched individually for the k nearest neighbors of a state for each action and the q-value is calculated by the arithmetic mean over the k stored values.

3 Queuing Simulation as Case Study

3.1 Simulation Description

We consider 4 different scenarios from a typical queuing simulation at a service desk. In the simple model, the agent can change the configuration o nce i n each time window between 0 and 5 open cash registers, while in the more complex model 10 changes can be made per time window and it is possible to open up to 7 cash registers simultaneously. The service time for each customer is 0.25 hours, and the staff budget is 5 hours in both cases. In the stochastic case, the arrival times are sampled from a Poisson distribution where we have peaks in the later morning and in afternoon. The patience p of the customers is sampled from a uniform distribution between 3 and 5, and the service time is sampled from a normal distribution with mean 0.25 hours and standard deviation $\sigma = 0.05$ hours. In the more complex case, the agent can make a decision and redeploy staff 10 times per hour. Therefore, the complexity increases slightly. This simulation is designed in such a way that in the deterministic case, 60 customers visit the bank per day, and 15 staff hours would theoretically be required to serve for all customers. However, there are far too few resources. The state-space of the agent consists of the time of day, the length of the queue, and the number of open desks. One episode is divided into ten-time units. A reward of $+10$ is received for a completed request and -10 for an abort.

3.2 Experimental Results

Over 10k episodes were simulated for all 4 scenarios. Mean and standard deviation were determined over 20 independent runs. Known approximation methods from the literature were taken up as a comparison. These were configured to the best of our k nowledge. The results can be seen in Figure 2.

In the simple case, classical DQN can adapt very well to both the deterministic and the stochastic variants. Linear regression, even in the simple case, is able to provide useful state approximation only at a very late stage. In the more complex case, the kNN variants lead very quickly to a good policy, while other methods would probably require further and more costly adaptation of the design parameters. We can verify that the approximation by the kNN method works sufficiently well in both cases and that classical methods by state aggregation do not have sufficiently well approximation properties in connection with RL. Neural networks also provide a good performance, but were complex to configure in order to achieve a learning behavior.

4 Discussion and Conclusion

We observe that the state space approximation by *k*-NN is superior to classical simple methods. In performance comparison with DQN, the universal approximation property of neural networks can provide good performance here. A fairly big advantage of this method in the context of simulations is the robustness against proper ranges of values in the state representation and the necessary design parameters in the approximation itself. These are serious advantages in the development and actual usage as an optimization method. While in parametric approximation methods one has to find a very balanced dimensioning for a good performance, this is not necessary in the present case. Furthermore, the simulation variables can be taken directly and do not need to be specially prepared by some pre-processing. This is remarkable especially in the case of neural networks, where the performance can be very relevant mostly due to the size of the network and also the encoding of the information in the first l ayer. While the *k*NN approximation does not require any design parameters, for the other methods a complex evaluation process was necessary to show any learning behavior at all. Especially the choice of the neural network leaves only a narrow range between over and underfitting. Therefore, we see the advantages of the *k*-NN approximation in connection with classical RL algorithms especially in the parameter-free operation.

5 Future Work

Subsequently, we will look at how well this method works for more complex simulations with non-equidistant decision logic. We also want to extend our basic architecture to agent-based simulations that are not explicitly designed for use within an MDP. Another opportunity is to apply this non-parametric method with current policy gradient algorithms in the actor-critic design pattern. We expect it to be particularly well suited in the stochastic case.

References

[1] Brockman G et al. “Openai gym”. In: *arXiv preprint arXiv:1606.01540* (2016).

[2] Choo B et al. “Reinforcement Learning from Simulated Environments: An Encoder Decoder Framework”. In: *2020 Spring Simulation Conference (SpringSim)*. May 2020, pp. 1–12.

[3] Doltsinis S, Ferreira P, and Lohse N. “An MDP Model-Based Reinforcement Learning Approach for Production Station Ramp-Up Optimization: Q-Learning Analysis”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 44.9 (Sept. 2014), pp. 1125–1138.

[4] Greasley. A. “Architectures for Combining Discrete-event Simulation and Machine Learning”. In: *Proceedings of the 10th International Conference on Simulation and Modeling Methodologies, Technologies and Applications - SIMULTECH,* INSTICC. SciTePress, 2020, pp. 47–58.

[5] L C and Jean-François S. “Discrete Event Modeling and Simulation for Reinforcement Learning System Design”. In: *Information* 13.3 (2022).

[6] Lang S, Kuetgens M, Reichardt P, and Reggelin T. “Modeling Production Scheduling Problems as Reinforcement Learning Environments based on Discrete-Event Simulation and OpenAI Gym”. In: *IFAC-PapersOnLine* 54.1 (2021). 17th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2021, pp. 793–798.

[7] Mahadevan S and Theocharous G. “Optimizing Production Manufacturing Using Reinforcement Learning”. In: *Proceedings of the Eleventh International Florida Artificial Intelligence Research Society Conference*. AAAI Press, 1998, pp. 372–377.

[8] Mnih V et al. “Playing Atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602* (2013).

[9] Qu S, Wang J, and Shivani G. “Learning adaptive dispatching rules for a manufacturing process system by using reinforcement learning approach”. In: *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. Sept. 2016, pp. 1–8.

[10] Sutton RS and Barto AG. *"Reinforcement Learning: an Introduction"*. 2nd. The MIT Press, 2018.

[11] Waschneck B et al. “Optimization of global production scheduling with deep reinforcement learning”. In: *Procedia CIRP* 72 (2018). 51st CIRP Conference on Manufacturing Systems, pp. 1264–1269.