Optimal Policies for Reinforcement Learning Applied to User Scheduling Tabular Environments

Caio Brasil, Ingrid Cavalcante, Cleverson Nahum, Jasmine Araujo and Aldebaro Klautau

***Abstract*— User scheduling is a fundamental task in shared systems where multiple users or processes compete for limited resources. Its main objective is to allocate these resources efficiently while ensuring fairness, high performance, and adherence to quality of service (QoS) requirements. In this paper, we explore the use of Reinforcement Learning (RL) methods to address the user scheduling problem in a well-defined scenario. Our results show that when the problem is modeled as a fully observable finite Markov decision process (FMDP), deep reinforcement learning methods exhibit apparent training convergence. However, when compared to classical approaches such as Value Iteration, there remains noticeable room for policy improvement in these methods, making them suitable as baselines for further implementations.**

***Keywords*— Reinforcement learning, User scheduling, Baseline**

I. Introduction

User scheduling is a fundamental process in shared systems where multiple users or tasks compete for limited resources. Its primary goal is to manage resource allocation efficiently, ensuring fairness, maximizing performance, and meeting quality of service (QoS) requirements [1]. In contexts such as wireless communications, user scheduling determines which users are granted access to system resources at any given time and frequency based on factors like channel conditions and priority levels. For that, the Base Station (BS) can consider observing the context of, for example, User Equipment (UE) location, buffer occupancy, or channel indicators.

Basic scheduling algorithms like Round Robin distribute resources equally among all UEs [2], while others use the resource block and signal to noise ratio information to avoid accumulating data in the user‘s buffer [3]. However, in 5G networks, the scheduling process becomes significantly more dynamic and complex due to the increased number of users, services, and varying network conditions. This important complexity has driven research into the application of Machine Learning techniques for user scheduling. Among these, Reinforcement Learning (RL) stands out as a promising approach, offering a flexible and adaptive framework for handling the intricacies of modern scheduling challenges.

RL is a powerful paradigm within the field of artificial intelligence that enables machines to learn and make decisions in dynamic and uncertain environments. It can be applied in a wide range of topics, such as robotics, telecommunications, and Natural Language Processing, where it is used mainly with deep learning-based (deep RL) methods that are based on neural network models [4]. However, the principal issue with those techniques is their potential to obtain optimal policies, which is affected by the deadly triad [5] that impacts the convergence capability of such techniques. Related work on user scheduling using RL [6], [7], [8] compares their proposed techniques using state-of-the-art baselines. Still, they do not specify optimal solutions, claiming they are unfeasible in their highly complex scenario. In the absence of optimal solutions in highly complex scenarios, an alternative for assessing these methods' performance is comparing them with optimal baselines in scenarios with low complexity before scaling them for the actual network scenario. This would increase the confidence that the proposed technique and baselines can reach an optimal or near-optimal performance in controlled scenarios before coping with complex ones.

When there are controlled environments, for example, where the number of actions and states is finite, Finite Markov Decision Process (FMDP), and the dynamics are perfectly known, the use of tabular methods can achieve the optimal policy more efficiently than deep methods [9]. Thus, tabular methods can guide the creation and modeling of the deep ones, serving as a baseline in controlled test environments.

This paper presents a comprehensive analysis of the use of both tabular and approximated RL methods over one well-defined user scheduling problem, in which the number of states and actions is finite and its dynamics are fully known. For the tabular side, there will be evaluated the Value Iteration (VI), and for the approximated methods, the Deep Q_learning (DQN) and Proximal Policy Optimization (PPO). The VI was used for its ability to compute optimal policies, serving as a baseline for the other methods.

II. Environment Description

Similar to the communication system model adopted in [10], a downlink single-user massive MIMO system is used in Vehicle-to-Infrastructure (V2I) environment where it is assumed that there is a single cell with one BS and U UEs. The incoming traffic is stored in buffers located at the BS. There is one buffer per UE, each with the capacity to store B packets. The UEs and the BS are positioned in a $G \times G$ grid-world. The convention adopted for the UE or BS position in this grid is P(x, y) with $x \in \{0, 1, ..., G-1\}$ and $y \in \{0, 1, ..., G-1\}$.

In this problem, it is assumed that a single grid position can be simultaneously occupied by all UEs. The movement of the UEs follows a uniform distribution over the adjacent

Caio Brasil, Ingrid Cavalcante, e-mail: caio.brasil, ingrid.nascimento@itec.ufpa.br; Cleverson Nahum, Jasmine Araujo and Aldebaro Klautau, e-mail: cleverson, jasmine, aldebaro@ufpa.br; Belém, Universidade Federal do Pará.

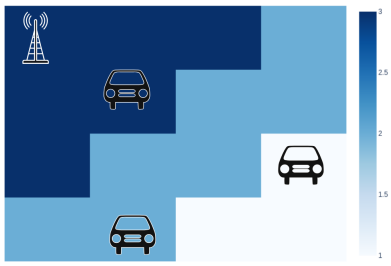

Fig. 1. Wireless environment with vehicles and signal strength gradients.

grid positions that are within one unit of distance and remain inside the grid boundaries. Channel quality is highest near the base station (BS) and degrades with increasing distance. Additionally, a fixed packet arrival rate is considered for all UEs at each time step in the environment. The Figure 1 exemplifies the problem formulation. It assumes that the spectral efficiency associated with the channel of a given user at time t coincides with the number of packets that can be sent by this user at time t so the scale of the heatmap corresponds to the number of packets that can be sent in a position.

Based on this problem specification, the RL environment representation consists of an observation space given by the buffer occupancy in the UEs and its position. Based on this information, the agent located at the BS will provide its action, which is the UE selected to provide the resources. With that, the problem's goal is to minimize the number of lost packets of the UEs, and the reward maximization sought by the agent during its learning is given by

$$R = \sum_{t=1}^{T_e} -10 \times L_t, \tag{1}$$

where L_t is the sum of lost packets of all UEs in a given time step t.

In this environment, the RL state is an element of the observation space's set. At each discrete time step, the system is in a state s which encodes the positions and buffer levels of all users. More specifically, for each user $i \in \{1, \ldots, U\}$, the state records their position (x_i, y_i) in a two-dimensional grid and their buffer occupancy $b_i \in \{0, \ldots, B\}$, where B is the maximum buffer size. So, the number of states that compose this environment is given by

$$S = ((G^2 - 1) \times (B + 1))^U. \tag{2}$$

To generate the optimal policy, some RL methods consist of analyzing all the environment's dynamics, which can be understood as the probability of the agent obtaining both some next state and its rewards given a pair of state and action. It is formalized with Markov Decision Process given by

$$p(s', r|s, a) = Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}, \tag{3}$$

where s' is the next state of some transition, r is the reward, s is the transition starting state, and a is the action taken.

However, for a controlled environment, the probabilities and rewards can be associated by the triplet (s', a, s). Thus, it gets two expressions that compose the environment dynamics: $p(s'/s, a)$ and $r(s, a, s)$. Normally, the structures used to store such dynamics are arrays; however, the larger number of states and actions can make loading such a kind of environment unfeasible. Thus, to allow the use of more complex environments, it is useful to analyze its sparsity, which can be understood as the ratio of null elements present in the probability's set.

To obtain the dynamics of the provided environment, the central assumption underlying the independence UE movements. This means that the future position of each user is independent of the movement choices of other users, allowing the global transition dynamics to be determined as the Cartesian product of each user's individual movement options.

An action a corresponds to scheduling one of the users to transmit data. The selected user's buffer is deterministically reduced by the number of packets that can be transmitted from their current position. After the action is applied, all users, including the one just scheduled, stochastically transition to new positions. Each user can either stay in place or move to one of the adjacent grid cells (up, down, left, or right), provided that the move stays within the grid boundaries and does not place the user at the base station's location. The possible new positions for user i are denoted by the set $\mathcal{M}_i$.

Under the independence assumption, each user's movement is uniformly random over their respective set of feasible moves. Therefore, the probability that user i moves to a specific new position $p_i \in \mathcal{M}_i$ is given by:

$$P(p_i \mid s) = \frac{1}{|\mathcal{M}_i|}. \tag{4}$$

Consequently, the joint probability of a specific combination of movements for all users $\mathbf{p} = (p_1, \ldots, p_{N_u})$ is the product of the individual probabilities:

$$P(\mathbf{p} \mid s) = \prod_{i=1}^{N_u} \frac{1}{|\mathcal{M}_i|}. \tag{5}$$

Once the new positions are determined, each user's buffer is updated based on the number of incoming packets at their new location. If the buffer exceeds its capacity B, the excess packets are considered lost. Then the reward associated with a transition from s to s' is calculated. So, the system then transitions to a new global state s' which encodes the updated positions and buffers of all users. The transition probability from state s to s' under action a is equal to the joint probability of the corresponding combination of user movements.

III. Overview of RL Algorithms

This section outlines the algorithms used in the study: Value Iteration (VI) as a tabular method using lookup tables, and Deep Q-Network (DQN) and Proximal Policy Optimization (PPO) as deep learning methods that employ neural networks and gradient-based updates.

A. Tabular RL

1) Value Iteration: The VI algorithm is a tabular method guaranteed to find optimal policies, the best actions to choose

in each state, in environments modeled as FMDPs with fully known dynamics [9]. Its original formulation is based on solving the Bellman equation for $V(s)$, as shown in Equation 6, but it can also be adapted to solve the Bellman equation for the action-value function. This equation captures the idea that the Q-value of an action in a given state is the expected sum of the immediate reward and the highest possible future reward, weighted by the probabilities of transitioning to each next state and receiving each reward. Here, γ is the discount factor, which determines how much the future rewards influence the current state's value.

$$V(s) = \sum_{s',r} p(s', r|s, a)[r + \gamma \times V(s')] \tag{6}$$

In the VI algorithm, first, the value function $V(s)$ is set to zero (or arbitrary values) for all states. Then, in each iteration, the value of every state is updated by taking the maximum expected return over all possible actions, considering the immediate reward and the discounted value of the next state. This process continues until the value function converges within a specified threshold θ. Once the optimal value function is obtained, the optimal policy, $\pi^*(s)$ is extracted by selecting, for each state, the action that maximizes the expected return. Its implementation is given by the Algorithm 1.

Algorithm 1 Value Iteration

Input: Environment, threshold θ
Initialize $V(s) \leftarrow 0$ for all $s \in \mathcal{S}$
repeat
  $\Delta \leftarrow 0$
  for each state $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s',r} P(s', r \mid s, a)\,[r + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end for
until $\Delta < \theta$
Output: Optimal value function $V(s)$
Derive policy: $\pi^*(s) \leftarrow \arg\max_{a \in \mathcal{A}} \sum_{s',r} P(s', r \mid s, a)\,[r + \gamma V(s')]$

It is important to note that this strategy exhaustively evaluates all possible state-action pairs to determine the optimal policy, which becomes infeasible in environments with continuous state or action spaces. Even when the state and action spaces are discrete and limited, the quality of the resulting policy remains highly dependent on the accuracy of the estimated environment dynamics. Nevertheless, in fully observable MDPs (FMDPs), the ability to compute optimal policies makes this approach valuable for modeling and benchmarking other strategies, serving as a reliable reference or guideline.

B. Deep RL

1) Deep Q-learning: The Deep Q-Network (DQN) algorithm extends Q-learning by using a deep neural network to approximate the action-value function $Q(s, a)$ [11], making it feasible to learn in high-dimensional or continuous state spaces. However, it only allows a discrete space of actions. At each time step, the agent selects actions via an ϵ-greedy policy, observes transitions, and stores them in an experience replay buffer. Learning is performed by sampling mini-batches of past experiences and minimizing the temporal difference (TD) error between predicted Q-values and target values. This allows the agent to learn efficiently from a broader distribution of data.

DQN addresses key issues associated with the *Deadly Triad*, the instability that arises from combining function approximation, bootstrapping, and off-policy learning [12]. The use of *experience replay* mitigates harmful correlations in sequential data, helping to break the feedback loop between Q-value estimates and the data they influence. The *target network*, which is updated separately from the main Q-network, stabilizes bootstrapping by providing a slowly changing target. The loss function used to train the neural networks is the mean squared TD error between the predicted Q-values and the target values, given in Equation 7.

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right] \tag{7}$$

where θ^- corresponds to the target network weights, θ is the prediction network weights, and (s, a, r, s') is a sample of the transition stored in the buffer for the experience replay.

2) Proximal Policy Optimization (PPO): Proximal Policy Optimization (PPO) is a reinforcement learning algorithm designed to improve the stability of policy gradient methods by preventing large, destructive policy updates.

The main idea is to optimize a surrogate objective that ensures the new policy π_θ does not change too much from the old policy $\pi_{\theta_{\text{old}}}$. This is done by using the probability ratio:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, \tag{8}$$

where (s_t, a_t) are state-action pairs from collected trajectories. The advantage estimate A_t measures how good an action was compared to the expected performance.

Instead of maximizing $r_t(\theta)A_t$ directly, PPO uses a clipped surrogate objective to limit policy changes:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)A_t\right)\right], \tag{9}$$

where ϵ is a small constant (e.g., 0.2). This clipping prevents the new policy from moving too far from the old one.

Additionally, PPO includes a value function loss:

$$L^{\text{VF}}(\phi) = \mathbb{E}_t\left[(V_\phi(s_t) - R_t)^2\right], \tag{10}$$

to improve the critic, and often an entropy bonus to encourage exploration.

The total PPO loss combines these terms:

$$L(\theta, \phi) = -L^{\text{CLIP}}(\theta) + c_1 L^{\text{VF}}(\phi) - c_2 \mathbb{E}_t[\mathcal{H}(\pi_\theta(\cdot|s_t))], \tag{11}$$

where c_1 and c_2 are coefficients, and $\mathcal{H}$ denotes policy entropy. PPO minimizes this combined loss to balance policy improvement, value estimation, and exploration.

IV. Experiments and Results

In this paper, it is assumed that there is a single cell with one BS to serve three users. Also, it is adopted: $G = 4$ and $B = 3$, which gives a finite number of states using the Equation 2 equal to 216,000. Since the number of actions also is finite, the environment can be modeled as a Finite Markov Decision Process (FMDP). Also, the spectral efficiency allows the UE to send all its packets close to the BS and decreases the amount as the UE moves away. The incoming packets are fixed for all UEs and all grids and are equal to 2 for each time step.

The environment adopted has 216,000 states and 3 actions, so to store all the transitions, it would be needed over 130 billion array elements to store everything. However, not all transitions that are possible in the adopted environment are feasible, which means a presence of a large amount of null probabilities stored in that array. So, the strategy to avoid this sparsity problem was to consider only possible transitions, reducing the number to close to 40 million transitions to help the tabular RL method to compute its policy efficiently.

The experiments consist of running 3 RL methods described previously (VI, PPO, DQN), over the environment, then analyzing their performance and discussing their convergence. It was used Stable Baselines 3 [13] to implement the deep RL methods and a standard implementation of the VI. All methods used γ equal to 0.9. PPO and DQN were trained over a total of 4 million time steps. Some parameters of the deep RL methods are present at the Table I; the others were the Stable Baseline default values.

TABLE I

Algorithm Configuration Parameters

Algorithm	Parameters
VI	θ equals to 0.0001.
DQN	Buffer size equal to 1,000,000 with a batch size of 32 samples. Learning rate equals to 0.0001. Target network is updated after 2000 steps. Topology: one hidden layer with 64 neurons.
PPO	Buffer size equal to 60,000 with a batch size of 128 samples. Learning rate equals to 0.0003. Clip factor equals to 0.2. c_1 and c_2 are equal to 0.5 and 0, respectively. Topology: Both policy network and value network has one hidden layer with 64 neurons.

For the Deep Reinforcement Learning (Deep RL) methods, a comprehensive analysis was conducted to evaluate the training processes. The corresponding results are depicted in Figure 2 for the Deep Q-Network (DQN) training and in Figure 3 for the Proximal Policy Optimization (PPO) training.

Fig. 2. DQN learning process

Starting with the analysis of the reward curves, it is evident that the PPO algorithm achieved a stable reward performance around the 2 million training step mark. In contrast, the DQN agent demonstrated faster convergence in terms of reward stabilization, achieving relatively stable reward values after approximately 1 million training steps.

Regarding the evolution of the loss functions, distinct behaviors were observed between the two methods. For PPO, the loss decreased steadily over time, ultimately approaching a value close to zero after about 2 million training steps. This pattern reflects the typical behavior of the clipped surrogate objective in PPO, where successive policy updates become smaller and more refined, indicating a well-behaved and stable optimization process.

On the other hand, the DQN agent exhibited a different dynamic. The DQN loss function decreased and appeared to reach a form of convergence after approximately 600 thousand training steps. However, it is important to note that despite this early apparent convergence of the loss, the reward continued to improve well beyond this point, eventually stabilizing at around 1 million training steps. Furthermore, unlike PPO, the DQN loss did not approach zero. Some reasons for this behavior are related to characteristics of Q-learning algorithms, where the presence of function approximation, bootstrapping, and stochasticity in the environment and replay buffer prevent the loss from diminishing completely to zero.

After the Value Iteration (VI) computed its policy and the training of both Deep Reinforcement Learning (Deep RL) methods — DQN and PPO — was completed, all policies were evaluated under the same conditions: 2000 test episodes, each consisting of 100 steps.

The distribution of the mean cumulative rewards obtained by each method is depicted in Figure 4. From this violin plot, it is evident that the VI method achieved consistently higher rewards compared to both DQN and PPO. Although the deep RL methods demonstrate some degree of convergence after extensive training, their reward distributions remain lower and more dispersed compared to the VI solution. Complementing this observation, Figure 5 illustrates the per-episode difference

Fig. 3. PPO learning process

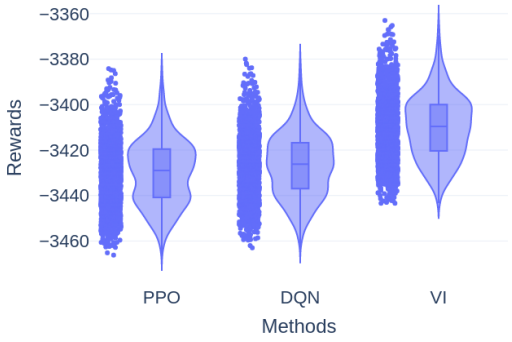

Fig. 4. Distribution of rewards.

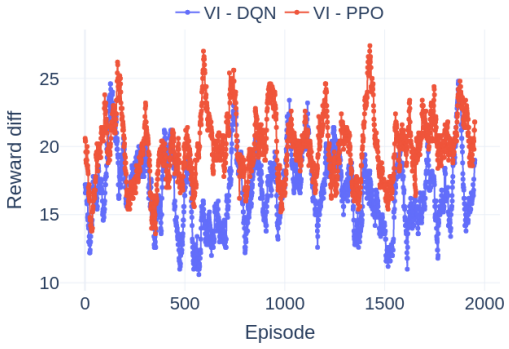

Fig. 5. Difference between rewards.

between the cumulative rewards obtained by VI and those obtained by DQN and PPO, respectively, across all 2000 test episodes. It is clear from the plot that VI consistently outperforms both deep RL methods, with the majority of the reward differences remaining positive and relatively stable across episodes.

In this context, an optimal algorithm like Value Iteration serves as a valuable performance benchmark. By assuming full knowledge of the environment's dynamics, VI computes the best possible policy, offering a theoretical upper bound for what learning-based methods can achieve. Its minimal need for parameter tuning also makes it well-suited for establishing baselines in well-defined environments, enabling meaningful comparisons and guiding the configuration of more complex methods before their deployment in real-world scenarios.

V. Conclusion

This paper aimed to evaluate the adaptability and effectiveness of three RL strategies under different levels of environmental uncertainty. Experimental results show that Value Iteration (VI) consistently outperforms the other methods across all conditions. Despite its limitations in more complex scenarios, VI remains useful during early development stages as a performance baseline. It can guide the tuning of more scalable methods like deep RL by providing a reference for optimal policy behavior.

Future work could explore more efficient implementations of VI for large state-action spaces and investigate how accurate a model must be for VI to produce near-optimal policies. This could help define its applicability in settings where only partial or noisy models are available.

Referências

[1] A. Mamane, M. Fattah, M. E. Ghazi, M. E. Bekkali, Y. Balboul, and S. Mazer, "Scheduling algorithms for 5g networks and beyond: Classification and survey," *IEEE Access*, vol. 10, pp. 51 643–51 661, 2022.

[2] X. Yuan and Z. Duan, "Fair round-robin: A low complexity packet schduler with proportional and worst-case fairness," *IEEE Transactions on Computers*, vol. 58, no. 3, pp. 365–379, 2009.

[3] P. Korrai, E. Lagunas, S. K. Sharma, S. Chatzinotas, A. Bandi, and B. Ottersten, "A ran resource slicing mechanism for multiplexing of embb and urllc services in ofdma based 5g wireless networks," *IEEE Access*, vol. 8, pp. 45 674–45 688, 2020.

[4] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[5] H. Van Hasselt, Y. Doron, F. Strub, M. Hessel, N. Sonnerat, and J. Modayil, "Deep reinforcement learning and the deadly triad," *arXiv preprint arXiv:1812.02648*, 2018.

[6] J. Mei, X. Wang, K. Zheng, G. Boudreau, A. B. Sediq, and H. Abou-Zeid, "Intelligent radio access network slicing for service provisioning in 6G: A hierarchical deep reinforcement learning approach," *IEEE Transactions on Communications*, vol. 69, no. 9, pp. 6063–6078, 2021.

[7] M. Polese, L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "ColO-RAN: Developing machine learning-based xApps for open RAN closed-loop control on programmable experimental platforms," *IEEE Transactions on Mobile Computing*, 2022.

[8] C. V. Nahum, V. H. L. Lopes, R. M. Dreifuerst, P. Batista, I. Correa, K. V. Cardoso, A. Klautau, and R. W. Heath, "Intent-aware radio resource scheduling in a ran slicing scenario using reinforcement learning," *IEEE Transactions on Wireless Communications*, vol. 23, no. 3, pp. 2253–2267, 2023.

[9] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[10] R. Aben-Athar, C. Nahum, D. da Silva Brilhante, J. F. de Rezende, L. Mendes, and A. Klautau, "User scheduling and beam-selection with tabular and deep reinforcement learning," *XL SIMPÓSIO BRASILEIRO DE TELECOMUNICAÇÕES E PROCESSAMENTO DE SINAIS*, 2022.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013. [Online]. Available: https://arxiv.org/abs/1312.5602

[12] A. Plaat, *Deep Reinforcement Learning*. Springer Nature Singapore, 2022. [Online]. Available: http://dx.doi.org/10.1007/978-981-19-0638-1

[13] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html