EUROPEAN WORKSHOP ON ON-BOARD DATA PROCESSING (OBDP2021), 14-17 JUNE 2021

SCHEDULING DOWNLINK OPERATIONS USING REINFORCEMENT LEARNING

Luca Romanelli, Alessandro Benetton, and Mattia Varile

AIKO - Autonomous Space Mission, Turin, Italy

ABSTRACT

Space operations are performed in a dynamic and complex environment, which exhibits non-deterministic action outcomes and where unexpected events may require human intervention. In this regard, tasks' scheduling and optimization of onboard resources are crucial points in the problem of enabling autonomous capabilities aboard satellites. Reasoning agents and autonomous decision-making systems represent valid approaches to such kind of problem. Nevertheless, these solutions require large efforts in defining consistent knowledge bases and models about the operative environment and about the agent itself.

Reinforcement Learning (RL) is currently one of the most compelling research fields in AI. Specifically, an RL algorithm allows agents to learn how to perform actions in an autonomous way through interaction with the surrounding environment. Bearing that in mind is possible to start exploring the advantages of such kinds of algorithms applied to the problem of autonomous space missions. Specifically, the objective of the research hereby presented is the implementation of an autonomous agent, which emulates an operating Earth Observation satellite, capable of scheduling downlink operations in advance, taking actions accordingly to its available resources and the priority of the data generated, aiming to optimize its tasks outcome at the same time.

Key words: Space missions; Reinforcement Learning; Scheduling; Optimization problems.

1. INTRODUCTION

In recent years the attention towards operations planning, has grown attention in space community due to the increasing complexity involved in space operations. The appearance of commercial players and the development of mega-constellations will increase the pressure on operators and the complexity involved in space mission management. A possible solution to this increasing complexity is to use advanced technology for automation, Artificial Intelligence (AI) in particular can be a powerful tool for tackling and possibly solve this type of problem.

In this framework is possible to introduce another element: Markov decision process (MDP), which is an effective approach to model the sequential decision problem to achieve a long-term goal. Reinforcement Learning (RL) has been developed as a promising approach to solve MDP problems, where the agent makes sequential decisions through continuous interaction with the environment. The ultimate goal of the agent is to find an optimal policy to maximize the cumulative reward. In RL, the mapping between state and action is stored in tabular form, which is impractical, especially for the large state space and continuous action space. Combined with some form of function approximator – such as Deep Neural Networks (DNN) – model-free Deep Reinforcement Learning (DRL) is capable of making intelligent sequential decisions in challenging environments.

In recent years, this has resulted in a powerful frameworks capable of achieving, and in some cases exceeds, human-level performances (Silver [1]). As demonstrated by Mnih [2] et al. the use of Neural Networks as a function approximator, set a new end-to-end approach to RL that greatly reduces the computational complexity involved in using other methods, and the capability of generalizing to a wide range of possible scenarios.

This approach led us to adapt the RL problem framework to the downlink operations scheduling problem to solve on board, as a first proof of concept with the clear goal of extending this to other use cases.

In this paper, we design an AI-based scheduling algorithm for managing downlink operations. We treat the problem as a combinatorial optimization problem. Given a sequence of packets to be downloaded, the agent has to select a subset of packets with maximum total priority to fill the downlink window such that the cumulative length of the packets does not exceed its capacity.

The work is structured as follows. The problem framework and the actual solutions are presented in section 2. Section 3 presents the packet scheduling problem for downlink operations using the Markov Decision Problem formulation on which a DRL-based algorithm is applied. Then section 4 presents a series of experiments performed within the simulation environment plus additional information regarding the deployment on space grade hardware. Finally, section 5 concludes the paper by presenting some final thoughts and extensions towards next steps.

2. BACKGROUND

2.1. Space systems downlink operations

An important part of space operations, in particular in Low Earth Orbit, is directly affected by downlink schedules. The efficiency and reliability of this operation is crucial to deliver the objectives of a space mission.

However, many uncertainty sources can directly impact this operation: the quantity of samples generated during scientific missions, atmospheric conditions, memory constraints, bandwidth constraints are just some of the aspects to consider for designing an effective downlink scheduler system.

As an example of this complexity, let's consider an observation satellite in LEO with a single grey-scale camera that collects geo-referenced images, only of particular areas of the Earth surface and only in particular lighting conditions.

The photos are generated in a stochastic way which is mainly affected by the satellite and Sun position. The complexity is even greater considering other variables such as:

- the limited amount of storage capabilities of the onboard computer,
- the different priority of the generated data (e.g.: high priority to images, low priority to meta-data and auxiliary data),
- the limited and variable downlink bandwidth.

This environment can be partially solved using a rule-based system, however his *randomness* and the variability of conditions not directly connected to the mission (e.g.: clouds on the ground station) can greatly increase the number of states to envision and to solve.

In this context a learning system can understand his environment acting accordingly without an explicit definition of the states to foresee.

2.2. Introduction to Reinforcement Learning

The founding principle of Reinforcement Learning is learn by interacting with the world. Humans and more generally every organism in the world is interacting with it counteracting in order to survive or, in other terms in order to maximize the expectancy of surviving.

This expectancy is also known, in the simplified framework of Artificial Intelligence, as a reward function r.

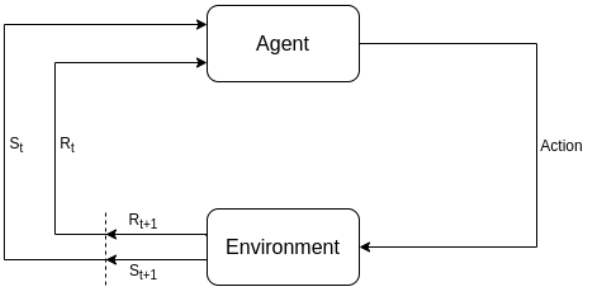


Figure 1. The Reinforcement Learning loop. An agent is interacting with the surrounding environment, changing his state (S_{t+1}) and getting a reward.

The final goal of the RL algorithm is to find an optimal policy π^* that maximizes the expected return – the cumulative reward the agent receives in the long run – which corresponds to the state value function V.

This process is described in Figure 1 (Sutton and Barto [4]). The agent is sensing as input the initial state S_t from the environment. On that basis, select an action, A_t. One time step later, in part as a consequence of its action, the agent receives a numerical reward, R_{t+1} and finds itself in a new state, S_{t+1}.

The agent continuously interact with the environment, and at a specific time step t the value function under the policy π is defined as:

$$V_\pi = \mathbb{E}_\pi[G^t] = \mathbb{E}_\pi[r^t + \gamma r^{t+1} + \gamma^2 r^{t+2} + ...] \quad (1)$$

where G represents the return and γ is a parameter between [0,1] called *discount factor*. The discount factor determines the present value of future rewards: as it approaches 1, the return objective takes future rewards into account more strongly.

As a practical example, an RL-based space lander needs to learn an optimal policy for controlling the engines and attitude system to avoid crashes on the ground or hard land. The agent is punished if the action taken leads to destruction or is rewarded in a successful landing.

2.3. Combinatorial optimization problems

Combinatorial optimization problems (COPs) can be frames as a method for searching the best element from a set of discrete elements; thus, in principle, any type of search algorithm or meta-heuristic can be used to solve them. Typical COPs are the traveling salesman problem (TSP), the minimum spanning tree (MST), and the knapsack problem.

The drawback of this class of methods are that: generic search algorithms are not guaranteed to find an optimal

solution and they are not guaranteed to run fast (in polynomial time).

We treated our problem as one of these combinatorial problems, the Knapsack Problem (KP). KP is a combinatorial optimization problem that aims to maximize the value of the items contained in a knapsack subject to a weight constraint. There are a few versions of the problem in the literature. We considered two of them as they are closely related to our scheduling problem.

In the binary (0-1) Knapsack problem, only one copy of each item is available, i.e., the agent can add items to the knapsack only once. Given a set of items, the goal is to determine the quantity of each item to include in a collection, so that the total weight is less than or equal to a given limit and the total value is as large as possible. Here, the complete set of items must be known a priori, and for each item, the weights and values are given. The problem has been well studied and is typically solved by dynamic programming approaches or by mathematical programming algorithms such as branch-and-bound.

The online version of the KP is stochastic; each item appears individually with a certain probability and must be either accepted or rejected by the agent. The goal here is the same as in traditional knapsack problems: to maximize the value of the items in the knapsack while staying within the weight limit, although it is more challenging because of the uncertainty about each available item.

3. DESIGN

In this section, we present our design choices for online scheduling with Reinforcement Learning. We approached the two combinatorial optimization problems described in the previous section, the binary KP and its online version, using instead a RL-based framework. We formulate the problem and describe how to represent it as an RL task by formalizing it as a Markov Decision Process (MDP). We then outline our RL-based solution, building on the algorithms described in the previous section.

3.1. The RL-based scheduling model

The goal is to design an agent capable of optimizing downlink efficiency by maximizing the priority of packets downloaded during periods of ground station (GS) visibility. These two main objectives relate mainly to the priorities and length of the packets to be downloaded and the available downlink capacity.

To address the problem in terms of RL, we have to transform the goal into a problem formulation and then formalize the problem itself as an MDP (Markov Decision Process).

We can think of the problem as an episodic task by making some assumptions. First, there is a collection phase in which the satellite collects all the data during its trajectory before reaching GS visibility. When a GS is reached, the agent starts performing actions, selecting a particular set of packets scheduled for the downlink to optimize the throughput and downlink efficiency. If the agent can no longer select any packet due to the downlink capacity or no more are available, the episode ends, and the loop repeats.

3.2. Problem formalization: the Markov Decision Process

State space

Since the problem is treated as an MDP, there is the implicit assumption of full observability of the environment. Considering the figure 2, the environment is internally characterized by the memory buffer containing the packets collected during the satellite's trajectory, and each packet is characterized by its length and priority. A possible representation of the state used by the agent to select actions could be a vector containing at each timestep:

- The priority and length of the *i-th* packet selected for the downlink at time-step t.
- The total amount of data — [KB] — stored in the satellites' memory and the relative sum of priority.
- The residual downlink capacity.

In the online version of the problem, the agent do not have the overall knowledge about the stored packets and the total values of length and priority. Data arrive in an online fashion, and the agent can only accept or reject the generated packet. This can happen, for example, because the data is generated continuously, even close to a downlink operation. Another reason could be the inability to prepare a buffer of packets to download just before the downlink. Then, the state representation we chose is a vector containing:

- the current priority and length of the packet to be scheduled
- current amount of data scheduled and the maximum downlink capacity

Action space

A possible space of actions could be any subset of the N packets stored in memory. However, in this way the dimension of the action space may become too large. Instead, we can allow the agent to perform more than one action at each "scheduling timestep" (the time does not elapse, and the agent continuously makes scheduling decisions until an invalid action is taken or there are no more packets inside the memory buffer):

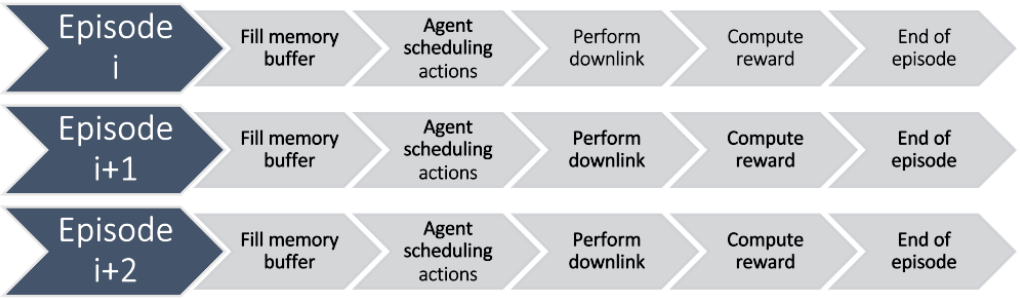


Figure 2. Scheduling model: environment-agent interaction

$$a_i = \begin{cases} 1, & \text{packet } i \text{ will be scheduled for download} \\ 0, & \text{packet } i \text{ will not be scheduled} \end{cases}$$

An invalid action is an attempt to schedule a packet does not fit inside the current downlink operation. We used the same action space for both problems. This is the only choice in the online version because we can only accept or reject a packet generated in a specific timestep. We could have used a larger action space for the offline case with an action for each packet. Still, we used a binary action space here for simplicity and allowed the agent to go through the entire sequence and decide to schedule or not a specific packet.

Reward function

To define an appropriate reward function, it is necessary to consider the agent's goal. In this case, a primary goal is to maximize the priority of downloaded packets along an episode. For this reason, we defined the reward function as described below:

$$r = \begin{cases} p_i, & \text{if packet } i \text{ is scheduled} \\ 0, & \text{otherwise} \end{cases}$$

where p_i is the priority of the i-th packet. In the offline case, to optimize the downlink efficiency and exploit it as much as possible, the agent also receives a negative reward proportional to the residual downlink capacity when it takes an invalid action or when no more packets to be scheduled are available, and the episode ends. In the online problem instead a negative reward is given to the agent when it tries to schedule a packet that exceeds the downlink capacity.

3.3. Agent design

For choosing the appropriate RL algorithm to train in this scenario, there are several aspects to consider. First, the nature of the task: as described in the previous section, the problem is seen as an episodic task. Then the agent can use the discounted reward setting to learn the best policy. When thinking of training the agent for a large number of episodes, the number of states that the agent will encounter is not known a priori, and it could become huge. For this reason, an approximation method for learning the value function or policy is needed. The best choice, in this case, is to use a deep neural network to have the possibility to learn the best features during training without constructing them by hand. Another thing to consider is whether the agent should learn the value function or the policy, or both. The final issue that matters for the choice of algorithm is the type of action space, hence the distinction between the discrete and the continuous case.
We used two different RL agents to solve both our problems, which belong to two families of algorithms: value-based and policy-based algorithms, respectively.

Value-based: DQN Agent

Considering all these aspects, we chose the Deep Q-Learning algorithm (Mnih et al. [3]) as a first solution for the scheduling problem related to the Binary KP. It approximates the Q-value function using a deep Q-net (DQN), consisting of an input layer proportional to the observation space, a certain number of hidden layers, and a linear output layer with one unit for each discrete action.
When introducing function approximation and neural networks in particular, we need to have a loss to optimize. For our implementation, we used the mean-squared TD-error, i.e., the mean squared error of the predicted Q-value and the target Q-value:

$$Loss = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right)^2\right] \quad (2)$$

The core idea behind the DQN algorithm is a technique called experience replay (Mnih et al. [2]), where the agent's experience is stored at each time step $e_t = (s_t, a_t, r_t, s_{t+1})$. During the inner loop of the algorithm, Q-learning updates, or minibatch updates, are applied to experience samples drawn randomly from the pool of stored samples. After performing the experience replay, the agent selects and executes an action according to an epsilon-greedy policy. Each step of the experience

is potentially used in many weight updates, allowing for greater data efficiency. In addition, randomizing the samples breaks their correlations and then reduces the variance of the updates. The second key idea behind this algorithm is the use of a separate network for estimating the target. This target network has the same architecture as the function approximator but with frozen parameters updated every N iterations, leading to a more stable training. We used the DQN algorithm to solve the scheduling problem when the overall content of the packets is known. We trained the agent for many episodes to generate a different set of packets and downlink operations and generalize as much as possible when the agent faces new data to schedule.

Policy-based: PPO Agent

The DQN is an action-value method cause it learns the values of actions and then selects actions based on its estimated action values; its policy would not even exist without the action-value estimates. Policy-based methods instead learn a parameterized policy that can select actions without consulting a value function. A value function may still be used to learn the policy parameter but is not required for action selection: the policy parameters are learned based on the gradient of some scalar performance measure with respect to the policy parameter. All methods that follow this general schema are called policy gradient methods, whether or not they also learn an approximate value function. Methods that learn approximations to both policy and value functions are called actor-critic methods, where 'actor' refers to the learned policy, and 'critic' refers to the learned value function, usually a state-value function. There are some main advantages when using a parameterized policy that has to be learned:

- Policy-based methods have better convergence properties.
- Policy gradients are more effective in high-dimensional action spaces (so we can use them in continuous action spaces).
- Policy gradients can learn stochastic policies due to the policy's definition, where it outputs a probability distribution over actions.

Considering the online scheduling problem and the ability of policy-gradient algorithms to learn stochastic policy, we train a PPO (Proximal Policy Optimization, Schulman et al. [6]) agent to solve this problem. It is an actor-critic algorithm, where the actor is responsible for selecting the actions, and the critic network has to criticizes the actions taken by the actor (figure 3).

4. SIMULATION RESULTS

In this section, numerical results are presented to evaluate the performance of the proposed RL algorithms. All

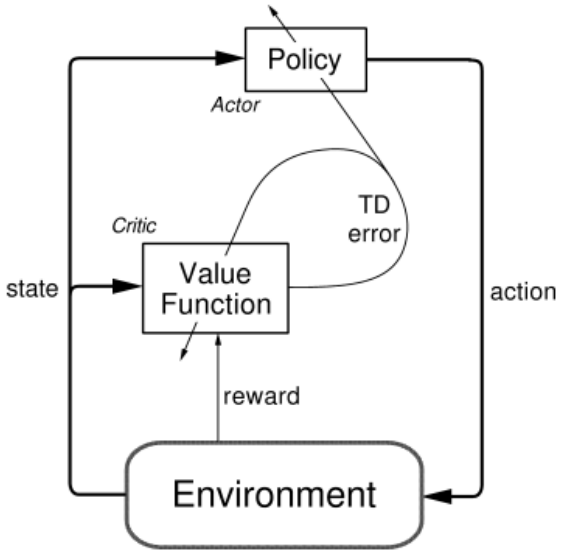


Figure 3. The actor-critic architecture

simulation results were obtained with Python and Tensorflow. For the environment implementation, we used Openai-Gym (Brockman et al. [7]), one of the most common toolkits for reinforcement learning research, which provides an interface that is compatible with almost all RL frameworks. Moreover, we compared the proposed algorithms with some baselines.

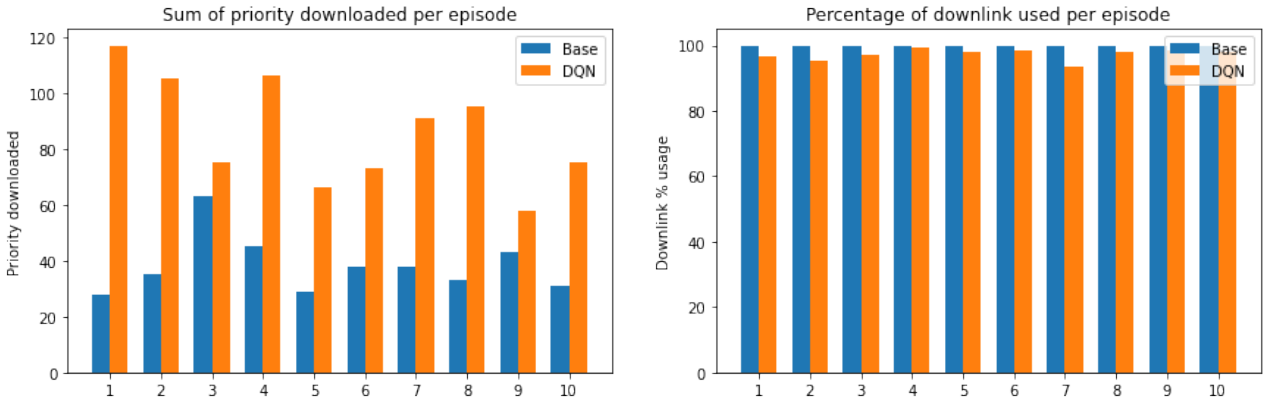
4.1. Simulation Setting

The environment setup is very similar for both the problems we solved. For the offline case, the buffer containing the data to be scheduled for downlink is set to 100 packets with random values for length and priority. The downlink capacity is a random value between two extremes (in terms of the quantity of data the satellite can download to the ground). In the online version, there is a window of 50 randomly generated packets in an online fashion.

Concerning the learning algorithm, the same architecture was used for all agent networks. It consists of 2 hidden layers with 64 units each. The rectified linear unit (ReLU) was used as the activation function. The discount rate γ was set to 0.99 during the training. The learning rate was set to $3\text{x}10^{-4}$, and the Adam optimizer was used for gradient descent.

4.2. Performance Evaluation

We used some baselines to do some benchmarks for evaluating the performances of the trained agents in solving both tasks. In the offline case, we used a simple heuristic algorithm that schedules packets after sorting them with respect to their priority. In addition, we



(a) Sum of downloaded priority

(b) Percentage of used downlink capacity

Figure 4. DQN-Baseline comparison in the offline case

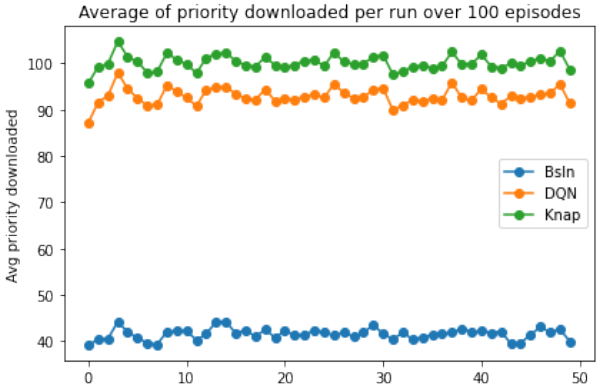


Figure 5. Average of results after 50 runs with 100 episode each in the offline case. The upper bound represents the optimal solution found by the dynamic programming approach.

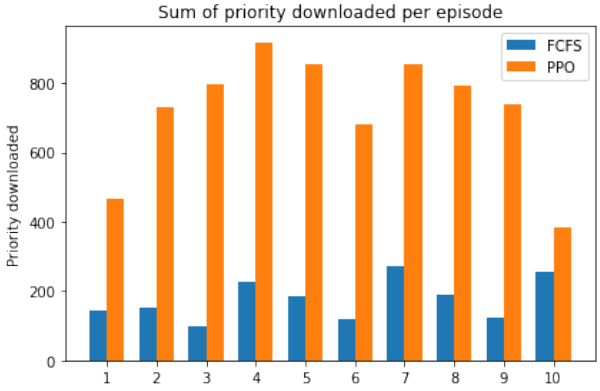


Figure 6. PPO-Baseline comparison, online case

used an optimal solution based on dynamic programming to have a measure of the maximum value of priority that can be obtained within a specific sequence of packets. The comparison was made by comparing the different agents/algorithms on the same data buffer in each episode. Figure 4 shows the results after running the DQN agent and the baseline after ten episodes. We can see that both exploit almost all of the downlink capacity, but in terms of total downloaded priority, the DQN agent performs quite better.

Figure 5 instead shows the average results over 50 runs with 100 episodes each. We can notice how the DQN agent approaches the optimal value found by the dynamic programming solution, bearing in mind that its recursive structure took a lot of computational time in some episodes to find a solution.

In the online version of the problem, we didn't have a greedy solution to compare with. So we used one of the most simply scheduling algorithms, a First Come First Service approach. Figure 6 shows the results after running the DQN agent and the baseline after ten episodes. We can notice how the PPO agent beats the baseline algorithm in each episode. Again, we averaged the results over 50 runs with 100 episodes each to have a more robust benchmark (Figure 6).

4.3. Deployment on embedded devices

The deployment process requires the adaptation of the original agent networks for the target embedded device. This is commonly accomplished by using Neural Network optimization framework such as Tensorflow Lite or Intel Openvino.

The commonly used pipeline usually involve many steps. Tensorflow APIs are generating, during the training process, a model (in the form of a single or multiple files) composed by an architecture specification file and an associated weights file. This model file is than optimized using a TFLite converter which reduces the model memory footprint and inference time (by performing *network quantization* [5] and *connection pruning*).

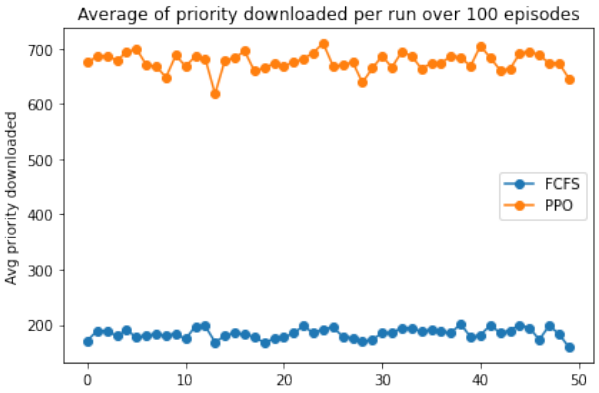


Figure 7. Average of results after 50 runs with 100 episode each, where the PPO agent is compared with the FCFS approach in the online case.

A first version of an optimized tflite version of the neural network has been produced. The estimated footprint on memory is lower than 100KB and the low number of model parameters is compatible with the deployment on multipurpose chips such as CPU/micro-controllers. This induces us to predict that the model will be compatible with space-grade ready devices. In future we will particularly focus our research on deployment on embedded devices.

5. CONCLUSIONS AND FUTURE WORK

This paper proposes an RL solution to the packet scheduling problem in downlink operations, where the goal is to maximize the throughput and the efficiency of such operations. We formalize the optimization problem as an MDP model. Value-based and policy-based deep reinforcement learning algorithms are applied to solve both offline and online cases. We demonstrate the effective ability of RL agents to learn how to solve such optimization problems.

Future improvements will consist in adding complexity to this scenario, for example, by adding some additional randomness to the downlink operations. Finally, as possible next steps, we will evaluate the application of the RL solutions to different types of optimization problems. Specifically, the online solution to the problem seems to be promising.

Finally, we plan to explore the deployment of the model on an embedded architecture, simulating a more realistic operational scenario

REFERENCES

[1] Silver, D., Huang, A., Maddison, C. et al. Mastering the game of Go with deep neural networks and tree search. Nature 529, 484–489 (2016). https://doi.org/10.1038/nature16961

[2] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning.

[3] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540):529.

[4] Sutton, R. S. and Barto, A. G. (2018), Reinforcement Learning: An Introduction , The MIT Press.

[5] Yunchao Gong, L. Liu, Ming Yang, Lubomir D. Bourdev (2014), Compressing Deep Convolutional Networks using Vector Quantization.CoRR, abs/1412.6115.

[6] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov: Proximal Policy Optimization Algorithms. https://arxiv.org/abs/1707.06347

[7] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, Wojciech Zaremba: OpenAI Gym https://arxiv.org/abs/1606.01540