Applicability and Challenges of Deep Reinforcement Learning for Satellite Frequency Plan Design

Juan Jose Garau Luis, Edward Crawley, Bruce Cameron
Massachusetts Institute of Technology
77 Massachusetts Av. 33-409
Cambridge, MA 02139
{garau, crawley, bcameron}@mit.edu

Abstract—The study and benchmarking of Deep Reinforcement Learning (DRL) models has become a trend in many industries, including aerospace engineering and communications. Recent studies in these fields propose these kinds of models to address certain complex real-time decision-making problems in which classic approaches do not meet time requirements or fail to obtain optimal solutions. While the good performance of DRL models has been proved for specific use cases or scenarios, most studies do not discuss the compromises of such models.

In this paper we explore the tradeoffs of different elements of DRL models and how they might impact the final performance. To that end, we choose the Frequency Plan Design (FPD) problem in the context of multibeam satellite constellations as our use case and propose a DRL model to address it. We identify six different core elements that have a major effect in its performance: the policy, the policy optimizer, the state, action, and reward representations, and the training environment. We analyze different alternatives for each of these elements and characterize their effect. We also use multiple environments to account for different scenarios in which we vary the dimensionality or make the environment non-stationary.

Our findings show that DRL is a potential method to address the FPD problem in real operations, especially because of its speed in decision-making. However, no single DRL model is able to outperform the rest in all scenarios, and the best approach for each of the six core elements depends on the features of the operation environment. While we agree on the potential of DRL to solve future complex problems in the aerospace industry, we also reflect on the importance of designing appropriate models and training procedures, understanding the applicability of such models, and reporting the main performance tradeoffs.

The motivation behind the use of DRL-based decision-making agents for satellite communications originates mainly from the forthcoming need to automate the control of a large number of satellite variables and beams in real-time. Large flexible constellations, with thousands of beams and a broad range of different users, will need to autonomously reallocate resources such as bandwidth or power in real-time in order to address a highly-fluctuating demand [8]. However, the new time and dimensionality requirements pose a considerable challenge to previously-adopted optimization approaches such as mathematical programming [9] or metaheuristics [10][12]. In contrast, due to the training frameworks common in ML, algorithms, DRL has the potential to meet these operational constraints [13].

Despite the positive results of DRL for satellite communications and many other fields, the majority of applied research studies mostly focus on best-case scenarios, provide little insight on modelling decisions, and fail to address the operational considerations of deploying DRL-based systems in real-world environments. In addition to the inherent reproducibility problems of DRL algorithms [14], recent studies show the drastic consequences of not properly addressing phenomena present in real-world environments [15] such as non-stationarity, high-dimensionality, partial-observability, or safety constraints.

As mentioned, high-dimensionality and non-stationarity are especially important in satellite communications. Mega constellations are already a reality, and they require models that adapt to orders of magnitude of hundreds or thousands of beams. Studies like [4,5,7] validate the proposed DRL models for cases with less than 50 beams, but obviate the performance in high-dimensional scenarios. In addition, user bases are becoming volatile and their demand highly-fluctuant. Relying on models that assume static user distributions could be detrimental during real-time operations. Therefore, it is crucial that applied research studies also discuss how their models behave against user non-stationarity and how they propose to address any negative influences.

One of the challenges of upcoming large constellations is how to efficiently assign a part of the frequency spectrum to each beam while respecting the interference constraints present in the system. This is known as the Frequency Plan Design (FPD) problem. Since users’ behavior is highly-dynamic, new frequency plans must be computed in real time to satisfy the demand. Given these runtime and adaptability requirements, in this paper we propose looking into a DRL-based solution
Table 1. Principal Reinforcement Learning parameters and their symbols.

| Symbol | Parameter |
|--------|-----------|
| $s_t$  | State of the environment at timestep $t$ |
| $a_t$  | Action taken by the agent at timestep $t$ |
| $r_t$  | Reward received at timestep $t$ |
| $S$    | Set of possible states |
| $A_t$  | Set of possible actions at state $s_t$ |
| $A(s_t)$ | Set of available actions at state $s_t$ |
| $s_T$  | Terminal state |
| $\pi$  | Policy |
| $\pi^*$ | Optimal policy |
| $\pi(a_t|s_t)$ | Probability of action $a_t$ given state $s_t$ |

In simple environments, the most effective policy is usually given by a table that maps each unique state to an action or probability distribution over actions. However, when the state and/or action spaces are large or continuous, using a tabular policy becomes impractical. Deep Reinforcement Learning (DRL) addresses such cases by substituting tabular policies for neural network-based policies that approximate the mapping from states to actions. These policies are then updated following Deep Learning algorithms, such as Stochastic Gradient Descent and backpropagation. DRL has shown significant results in a wide variety of areas like games [18], molecular design [19], or internet recommendation systems [20].

There are six elements that jointly drive the performance of a DRL model:

1. **State representation.** We want to look for state representations that capture as much information of the environment as possible and can be easily fed into a neural network. In the case of vision-based robots, that might be simply RGB images from cameras. In other contexts, the process sometimes involves more complex state design strategies. It can be also referred as state space.

2. **Action space.** Although flexibility matters, reducing the action space is generally beneficial to the algorithm. While, for instance, videogames’ actions are straightforward, other environments might require changes to the action space, such as discretization or pruning. It can be also referred as action representation.

3. **Reward function.** The reward function is relevant during the training stage of the algorithm, and should also be informative with respect to the goal of the agent. There might be more than one reward function that correctly guides the learning, although there is usually a performance gap between classes of strategies: constant reward, sparse reward, rollout policy-based reward, etc.

4. **Policy.** In DRL the policy is mainly represented by the neural network architecture chosen. DRL policies commonly include fully-connected layers in addition to convolutional and/or recurrent layers. Each of these layer classes includes multiple subclassifications to consider.

5. **Policy optimization algorithm.** There are many algorithms that rely on completely different design choices. In model-free DRL, alternatives include policy gradient methods, Q-learning, or hybrid approaches.

6. **Training procedure.** The training dataset and training environment constitute an important part of the performance. We want to reflect all phenomena that might take place during testing and make sure that the training procedure guarantees a robust performance. Sometimes, however, the possibilities of interaction with real environments are limited and models must resort to data-efficient strategies.

2. **Deep Reinforcement Learning**

   Deep Reinforcement Learning is a Machine Learning subfield that combines the Reinforcement Learning (RL) paradigm [16] and Deep Learning frameworks [17] to address complex sequential decision-making problems. The RL paradigm involves the interaction between an agent and an environment that follows a Markov Decision Process, and is external to the agent. This interaction is sequential and divided into a series of timesteps. At each of these timesteps, the agent observes the state of the environment and takes an action based on it. As a consequence of that action, the agent receives a reward that quantifies the value of the action according to the high-level goal of the interaction (e.g., flying a drone or winning a game). Then, the environment updates its state taking into account the action from the agent.

   The interaction goes on until a terminal state is reached (e.g., the drone lands). In this state, the agent can not take any further action. The sequence of timesteps that lead to a terminal state is known as an episode. Based on the experience from multiple episodes, the objective of the agent is to optimize its policy, which defines the mapping from states to actions. The agent’s optimal policy is the policy that maximizes the expected average or cumulative reward over an episode. This policy might be stochastic depending on the nature of the environment. Table 1 summarizes the parameters defined so far and includes the symbols generally used in literature.
which is twice the amount of reuse groups (we consider left-handed and right-handed circular polarization available for each beam). For each frequency group, there is an equal number of slots and polarization, and row 2 corresponds to reuse group 1 and slot 6, and needs a bandwidth of 3 slots – it therefore occupies slots 6, 7, and 8 in group 1. At time instant $t_2$, this beam undergoes a handover operation from satellite 1 to satellite 2. When this occurs, the beam is assigned the same group 1 and slot 6 in satellite 2 and frees them in satellite 1. While we could change the frequency assignment during the handover, this is not always possible due to various factors. As a consequence, a safe strategy is to make sure that, at the moment beam $b$ switches from satellite 1 to satellite 2, slots 6, 7, and 8 in group 1 are available in satellite 2.

**Constraints**

We identify two types of constraints. On one hand, we do not assume a specific orbit and therefore take into account that handover operations might occur. The frequency plan must account for handover constraints, which entail that some beams assigned to the same frequency group cannot overlap in the frequency domain because they are powered from the same satellite at some point in time. We define this type of constraints as intra-group constraints.

To better understand intra-group constraints and following the example in Figure 3, let’s assume beam $b$ (green beam in the figure) is powered from satellite 1 at time instant $t_1$. This beam is assigned to frequency group 1 and slot 6, and needs a bandwidth of 3 slots – it therefore occupies slots 6, 7, and 8 in group 1. At time instant $t_2$, this beam undergoes a handover operation from satellite 1 to satellite 2. When this occurs, the beam is assigned the same group 1 and slot 6 in satellite 2 and frees them in satellite 1. While we could change the frequency assignment during the handover, this is not always possible due to various factors. As a consequence, a safe strategy is to make sure that, at the moment beam $b$ switches from satellite 1 to satellite 2, slots 6, 7, and 8 in group 1 are available in satellite 2.

On the other hand, we define the other type of restrictions as inter-group constraints. These relate to the cases in which beams that point to close locations might negatively interfere with each other. This situation is more restrictive, as those beams not only must not overlap if they share the same frequency group but also if they share the same polarization. For example, if beams $b_1$ and $b_2$ had an inter-group constraint and beam $b_1$ occupied slots 6, 7, and 8 in group 5, then, following our previous numbering convention, beam $b_2$ could occupy any slot but slots 6, 7, and 8 in groups 1, 3, 5, 7, etc. This interaction is represented in Figure 3.

**4. METHODS**

The objective of this paper is to design and implement a DRL-based agent capable of making frequency assignment decisions for all $N_B$ beams of a constellation, to evaluate its performance, and to understand its modelling tradeoffs. While there exist algorithms that could address all of the decisions simultaneously, this approach makes the problem challenging for dynamic cases with large values for $N_B$, $N_{FG}$, and $N_{FS}$. The total number of possible different plans is $(N_{FG}N_{FS})^{N_B}$, which poses a combinatorial problem that does not scale well.

As an alternative, we take a sequential decision-making approach in which, following the DRL terminology previously introduced, a timestep is defined as the frequency assignment for just one beam. We then define an episode as the complete frequency assignment of all $N_B$ beams. By doing this, the dimensionality of the problem reduces to $N_{FG}N_{FS}$ for each timestep.

The next step in our formulation involves the definition of the remaining problem-specific elements present in a RL setup, i.e., state, action, and reward. Generally, the application of DRL to domain-specific problems involves the empirical study of which representation for each element better suits the problem. However, it might be the case that no single representation outperforms the rest in all scenarios, and therefore the representation can be regarded as an additional hyperparameter that depends on the environment conditions. Most domain-specific DRL studies leave these considerations out.
out of their analyses. To highlight the importance of the representation selection, in this section we propose alternative representations for each element and study how they affect the overall performance of the model.

Regarding the action, two different action spaces are studied. These are pictured in Figure 4, which shows a scenario with $N_{FG} = 4$ and $N_{FS} = 4$. First, we consider directly choosing a cell in the grid as the action. This action space, which we define as Grid, consists of $N_{FG}N_{FS}$ different actions. The second action space is defined as Tetris-like and only contemplates five possible actions. In this space, a random frequency assignment is first made for a beam, i.e., we randomly choose a cell in the grid. Then, the agent is able to move it up, down, left, and right across the grid until the action new is chosen and a new beam undergoes the same procedure. Note that with this latter approach, episodes take a longer and random number of timesteps, since one beam can take more than one timestep to be assigned and there is no restriction on how many intermediate actions can be taken before taking action new. The advantage of this representation is that the action space is substantially reduced, which benefits the learning algorithm.

Next, we also consider two possible representations for the state space. In both, the state is defined as a 3-dimensional tensor in which the first dimension size is 1, 2, or 3, and $N_{FG}$ and $N_{FS}$ are the sizes of the second and third dimensions, respectively. We refer to a slice of that tensor along the first dimension as a layer. To better understand both representations, we consider the context of a certain timestep, in which we are making the assignment for one specific beam $b_k$, we have already assigned $k - 1$ beams, and $N_B - k$ beams remain to be assigned. In both representations the first layer stores which cells conflict with beam $b_k$, since they are “occupied” by at least one of the beams, among the $k - 1$ already assigned, that have some kind of constraint with $b_k$. In the case the action space is Tetris-like, the second layer stores the current assignment of beam $b_k$ – this is done regardless of the state representation chosen. Finally, the last layer serves as a lookahead layer and is optional. Whether it is included in the tensor or not defines the two state representations considered. This layer contains information regarding the remaining beams, such as the number $N_B - k$, or the amount of bandwidth that will be compromised in the future for beam $b_k$ due to the beams remaining to be assigned which have an intra-group or inter-group constraint with $b_k$. Figure 5 shows an example of what these three layers can look like. We therefore run simulations with and without the lookahead layer.

Note that our selection of state and action representations entails an additional benefit: we could start from incomplete frequency plans. There might be situations in which we do not want to entirely reconfigure a plan but just reallocate a couple of beams – due to moving users, handover reconfiguration, etc. Furthermore, adding or removing beams to the constellation would not be a problem either. In that sense, the model would be robust against changes in the beam placement.

Lastly, three alternative functions are considered to define the reward. For the three of them, if the action space is Tetris-like, the reward function is only applied whenever the action
Figure 4. Two action spaces considered: Grid and Tetris-like.

Figure 5. Possible layers of the state space.

new is chosen, otherwise the value \(-\frac{1}{N_{FG}N_{FS}}\) is given as a reward. This is done to avoid the agent finding local optima that consist in just moving a specific beam around without calling the action new.

Once a beam is assigned (i.e., any action is taken in the Grid action space or the action new is taken in the Tetris-like action space), the first reward function consists of computing the difference in the number of successfully already-assigned beams between the states at the current timestep and previous timestep. We consider a beam to be successfully assigned if it does not violate any constraint. The second alternative is to only compute the final quantity of beams that are successfully assigned once the final state \(s_T\) is reached and give a reward of zero at all timesteps before that. Finally, inspired by its use in other RL papers [21], the third reward function is Monte Carlo-based and uses a rollout policy that randomly assigns the remaining beams at each timestep. This is only done to compute the reward, the outcome of the rollout policy is not taken as agent’s actions. We define each of these strategies as Each, Final, and Monte Carlo (MC), respectively. Figure 6 shows a visual comparison between the three options considered.

Before discussing the results, in Table 2 we summarize the variations that we consider for each of the six elements introduced in Section 2: state, action, reward, policy, policy optimization algorithm, and training procedure. In this section we have described the variations for the first three, in the next section we address the remaining elements, since they specifically relate to the learning framework.

5. RESULTS

To test the presented framework, we simulate a scenario with \(N_B = 100\), \(N_S = 7\), \(N_{FG} = 4\), and \(N_{FS} = 20\). From a set of 5,000 beams based on real data, we randomly create a train and a test dataset, disjoint with respect to each other. For each episode of the training stage, we randomly select 100 beams from the train dataset. Then, during the test stage we randomly select 100 beams from the test dataset and evaluate the policy on those beams. Since the nature of the problem is discrete, we initially select Deep Q-Network [18] as the policy optimization algorithm and later in this section we compare it against a policy gradient algorithm. We use 8 different environments in parallel, this way we can share the experience throughout the training stage and have statistics over the results during the test stage. We start using a policy consisting of a Convolutional Neural Network (CNN) with 2 layers (first layer with 64 5x5 filters, second layer with 128 3x3 filters), then 2 fully-connected layers (first layer with 512 units, second layer with 256 units), and an output layer. The fully-connected layers can be commonly referred as Multi Layer Perceptron (MLP). ReLU activation units and normalization layers are used in all cases.

Full enumeration analysis

We do a full enumeration analysis and test each possible combination of action-state-reward representation for a total of 50k timesteps. At this point, we also focus on the training procedure and test which improvements there might be if we train the agent to do the assignment with 200 beams instead of 100, to make the task harder and to potentially obtain a more robust policy. In this latter case, the test evaluation is still carried out on 100 beams. In addition, we extend
the grid search to include the discount factor $\gamma$ \cite{16}, a DRL hyperparameter, and evaluate three possible values: 0.1, 0.5, and 0.9. A larger discount factor implies that the agent takes into account longer term effects of its actions, whereas lower discount factors are related to greedier policies.

The results of the enumeration analysis for the 72 different simulations are shown in Table\ref{tab:1} which contains the average number of successfully-assigned beams during test time (out of 100) for each combination of action, state, reward, discount factor $\gamma$, and number of training beams. The average is computed across the 8 parallel environments. For reference, we compare these results against a totally random policy, which achieves an average of 83.5 successfully-assigned beams.

It is observed that using the reward strategy Each leads to better outcomes, and using the Grid action space with the With state space does better on average. For our case, there is no apparent advantage in using reward strategies that delay the reward until the end of the episode or rely on rollout policies; individual timesteps provide enough information to guide the policy optimization. To better understand the impact of the modelling decisions, we also do several significance tests and further analyze the tradeoffs:

- There is no advantage in training with 200 beams instead of 100 (p-value = 0.08). Therefore, training with more beams does not make a more robust policy for our case. This is even less impactful when $\gamma = 0.1$ (p = 0.90), since the policy behaves greedily regardless of how many beams remain to be assigned. The strategy for placing the first 100 beams is similar both cases.
- The discount factor affects the performance of the policy (p = 10^{-6}), but there is no significant performance difference when using $\gamma = 0.1$ or $\gamma = 0.5$ (p = 0.83). The performance worsens when $\gamma = 0.9$. The learned policy relies more on a greedy behavior to be successful, which is consistent with other methods that have addressed the problem \cite{22}.
- With $\gamma \leq 0.5$, the Each reward strategy, and Grid action space, the state space affects the performance (p = 10^{-6}). It is better to include the lookahead layer.
- With $\gamma \leq 0.5$, the Each reward strategy, and Tetris-like action space, the state space affects the performance (p = 10^{-6}). It is better to include the lookahead layer.
- With $\gamma \leq 0.5$ and the Each reward strategy, using the Grid state space is a better alternative than using Tetris-like (p = 10^{-5}), since it supports more flexibility to make an assignment.
- Overall, the Tetris-like action space appears to be less sensitive to the reward strategy chosen, especially when comparing across the simulations using the Monte-Carlo reward strategy.

The lookahead layer plays an important role together with the Grid action space, since the agent has total freedom to place a beam, and therefore being informed of the remaining assignments is beneficial. In contrast, in the case of the Tetris-like action space, the agent is limited by the initial random placement of the beam, and therefore the information of what to come is not as useful.

**Scalability analysis**

Given the high-dimensionality of future satellite systems, we also analyze the impact scalability has on our framework and run new simulations, this time for 500 beams ($N_B = 500$), with $N_S = 7$, $N_{FG} = 16$, $N_{FS} = 80$, $\gamma = 0.1$, and the Each reward strategy, and compare all 4 combinations for the state and action representations. In this case we train each model for a total of 200k timesteps. These results are shown in Table \ref{tab:2} and can be compared against a totally random policy, which achieves 429.9 successfully-assigned beams.

The main conclusion of this analysis is that when the dimensionality of the problem increases, we can actually achieve a better outcome using the Tetris-like action space, as opposed to the 100-beam case. The Grid action space does even worse than random, which is due to the large number of actions (1,280 for this case) and an inappropriate exploration-exploitation balance. Overall, there is no impact in terms of the state representation in this example (p = 0.64), but if we focus on the Tetris-like action space, there is (p = 0.001); we still do better without the lookahead layer. These results prove that relying on a single representation for a specific problem might not be a robust strategy during operation. Understanding the limitations of specific representations is essential in order to make these systems deployable and, more importantly, reliable.

**Policy and Policy Optimization Algorithm**

Two of the key elements of a DRL model listed in Section\ref{sec:2} and Table\ref{tab:3} the policy and the policy optimization algorithm, remain to be studied for our problem. These heavily rely on the progress made by the Deep Learning and Reinforcement Learning research communities. To address them, we now extend the analyzes to compare the results of the Deep Q-Network (DQN) algorithm, and the CNN plus the MLP policy, both used in all simulations so far, against other alternatives.

In the case of the policy optimization algorithm, we choose the Proximal Policy Optimization (PPO) \cite{23} method, a Policy Gradient and off-policy algorithm, to be compared against DQN \cite{18}, a Q-learning and off-policy algorithm. PPO optimizes the policy on an end-to-end fashion, by doing...
gradient ascent on its parameters according to a function of the cumulative reward over an episode. Additionally, it clips the gradients to avoid drastic changes to the policy. DQN focuses on optimizing the prediction of the value of taking a certain action in a given state; it then uses these predictions to choose an action. Also, it stores all the agent’s experience and makes use of it over time by “replaying” past episodes and training on them. We refer to the original papers for a full description of each method. We use OpenAI’s baselines [24] to implement each method.

Regarding the policy network, we substitute the MLP for a 256-unit Long Short-Term Memory (LSTM) [25] network, which belongs to the recurrent networks class. We compare a policy constituted by CNN+MLP against a CNN+LSTM one to evaluate the impact of using a hidden state in our formulation. Recurrent neural networks take advantage of temporal dependencies in the data and therefore can potentially perform better in sequential problems.

Table 3 shows the results of using the different policies and the different policy optimization algorithms for 4 different cases: the 100-beam scenario using the Grid action space with both state representations, the 500-beam scenario using the Tetris-like action space with both state representations, and two additional 1,000-beam and 2,000-beam scenarios for the Tetris-like action space and the Without state representation. For each case the random policy successfully-assigns 83.5, 429.9, 799.7, and 1,137.7 beams on average, respectively. Since we have concluded the Tetris-like action space better suits high-dimensionality scenarios, we want to explore its performance in the thousand-beam range as well. In all cases the Each reward strategy and $\gamma = 0.1$ is used. All simulations belonging to the same scenario are trained for an equal number of timesteps.

In the scenarios with 1,000 beams or less, we can observe that there is no significant advantage in using PPO over DQN, although DQN consistently outperforms PPO in each scenario. Looking at the 2,000-beam scenario, however, the performance difference emerges. Note that the 1,000-beam and 2,000-beam simulations use the same values for $N_{FG}$ and $N_{FS}$. Therefore, in the 2,000-beam case, the performance worsening occurs during the assignment of the last 1,000 beams. During the last timesteps of such a high-dimensional scenario, having a prediction over multiple actions, as DQN does, proves to be a better approach, as opposed to PPO’s method, which relies on the single action that the policy outputs.

The same occurs if we compare the CNN+MLP policy (MLP in the table) against the CNN+LSTM policy (LSTM in the table). Given the greedy behaviour of the policy, having the LSTM’s hidden state does not offer any advantage when placing the last 1,000 beams in the 2,000-beam scenario. Although allowing more training iterations could help reducing the performance gap, these results prove that the choices of the policy and the policy optimization algorithm are especially important under certain circumstances. For the FPD problem in the context of megaconstellations, we need to care about the

---

### Table 3. Average number of successfully-assigned beams out of 100 in the test data. A random policy achieves 83.5.

| Reward | $\gamma$ | 100-beam training | 200-beam training |
|---------|----------|--------------------|-------------------|
|         |          | Grid With | Without | Tetris-like With | Without | Grid With | Without | Tetris-like With | Without |
| Each    | 0.1      | 99.8      | 95.6     | 92.0      | 95.8     | 98.9      | 97.9     | 87.5      | 98.2     |
|         | 0.5      | 98.9      | 96.2     | 90.5      | 96.6     | 97.6      | 94.4     | 80.8      | 90.9     |
|         | 0.9      | 96.6      | 93.1     | 81.5      | 66.9     | 95.2      | 93.4     | 70.2      | 66.6     |
| Final   | 0.1      | 47.5      | 53.9     | 68.8      | 67.4     | 23.8      | 20.9     | 52.5      | 45.5     |
|         | 0.5      | 46.6      | 56.0     | 39.9      | 70.1     | 21.1      | 21.2     | 67.0      | 69.9     |
|         | 0.9      | 54.9      | 67.2     | 71.9      | 63.6     | 25.8      | 22.6     | 60.0      | 67.1     |
| MC      | 0.1      | 19.5      | 17.6     | 45.2      | 45.2     | 2.5       | 2.9      | 50.9      | 34.0     |
|         | 0.5      | 9.1       | 3.0      | 67.9      | 36.1     | 2.6       | 1.9      | 41.1      | 20.2     |
|         | 0.9      | 2.2       | 2.6      | 37.8      | 47.5     | 2.2       | 3.2      | 34.6      | 24.1     |

### Table 4. Average number of successfully-assigned beams out of 500 in the test data using the Each reward function and $\gamma = 0.1$. A random policy achieves 429.9.

| Action and state representation | Number of successfully-assigned beams |
|---------------------------------|--------------------------------------|
| Grid and With                   | 320.1                                |
| Grid and Without                | 328.1                                |
| Tetris-like and With            | 461.6                                |
| Tetris-like and Without         | 478.9                                |

### Table 5. Average number of successfully-assigned beams in the test data when comparing different combinations of policy and policy optimization algorithm. The Each reward function, $\gamma = 0.1$, and $N_S = 7$ are used in all cases.

| Action and state representation | DQN MLP | PPO MLP | PPO LSTM |
|---------------------------------|---------|---------|----------|
| 100 beams, Random:             | 83.5    | 94.9    | 94.9     |
| $N_{FG} = 4$, $N_{FS} = 20$    |         |         |          |
| Grid, With                     | 99.8    | 94.9    | 94.9     |
| Grid, Without                  | 95.6    | 97.5    | 91.5     |
| $N_{FG} = 16$, $N_{FS} = 80$   |         |         |          |
| Tetris-like, With              | 461.6   | 460.5   | 470.0    |
| Tetris-like, Without           | 478.9   | 478.0   | 465.1    |
| 1,000 beams, Random:           | 799.7   | 936.0   | 957.0    |
| $N_{FG} = 20$, $N_{FS} = 100$  |         |         |          |
| Tetris-like, Without           | 962.3   | 1,137.7 |          |
| 2,000 beams, Random:           |         |         |          |
| $N_{FG} = 20$, $N_{FS} = 100$  |         |         |          |
| Tetris-like, Without           | 1,746.0 | 1,139.0 | 967.0    |
Table 6. Average number of successfully-assigned beams in the test data when comparing different imbalances between train and test data. The Each reward function, the DQN algorithm, the CNN+MLP policy, and $\gamma = 0.1$ are used in all cases.

| Action, State representation | Same demand | 2-times demand | 4-times demand |
|-----------------------------|-------------|----------------|----------------|
| 100 beams, $N_{FG} = 4$, $N_{FS} = 20$ | 99.8 | 84.9 | 52.6 |
| Grid, With | 95.6 | 84.4 | 65.3 |
| Grid, Without | 83.5 | 81.3 | 71.9 |
| Random | 461.6 | 453.0 | 249.0 |
| Tetris-like, With | 478.9 | 434.0 | 337.0 |
| Tetris-like, Without | 429.9 | 400.0 | 227.0 |

| Action, State representation | Same demand | 2-times demand | 4-times demand |
|-----------------------------|-------------|----------------|----------------|
| 500 beams, $N_{FG} = 16$, $N_{FS} = 80$ | 95.8 | 500-beam | 96.2% |
| Tetris-like, With | 461.6 | 453.0 | 249.0 |
| Tetris-like, Without | 478.9 | 434.0 | 337.0 |
| Random | 429.9 | 400.0 | 27.0 |

There are different alternatives to be robust in non-stationary environments. The first involves, after identifying the sources of non-stationarity, devising adequate training datasets that include episodes capturing those sources. In this case, it is essential to use a state representation that successfully incorporates information on the condition of each source. Secondly, we can make use of algorithms that refine the agent's actions under contingency scenarios. These algorithms could guarantee a most robust behavior at the expense of computing time, such as in [26], where a DRL model is combined with a subsequent Genetic Algorithm to maximize robustness. Finally, we can rely not only on a single agent but on an ensemble of agents that specialize on different scenarios and are trained in precisely different ways. In all cases, retraining the models over time from collected experience during operation is a good practice to capture any possible change in the environment.

**Impact on real-world scenarios**

The proposed models have shown a quantitatively good performance in the majority of the scenarios considered. On average, the agent successfully-assigns 99.8% of beams in the 100-beam case, 95.8% in the 500-beam case, 96.2% in the 1,000-beam case, and 87.3% in the 2,000-beam case. The main advantage of this approach is that evaluating a neural network is substantially faster than relying on other algorithms such as metaheuristics, while still showing a good performance. This aligns with what most applied DRL studies conclude. To correct not fulfilling the entirety of the constraints, we could have a subsequent “repairing” algorithm to address the conflicting beams, which could make up 1 to 15% of the total based on our numbers.

However, the main consideration behind these numbers is that they correspond to different models that use different state and action representations. Unlike many applied DRL studies, we have regarded representations as additional hyperparameters at every stage of our analyses. Given the importance of high-dimensionality in real-world operations, we have also focused on cases with hundreds to thousands of beams, as opposed to works that exclusively consider cases with less than 100 beams. We could not have extracted our conclusions without making these decisions for our analyses. When studying a specific DRL model with defined hyperparameters, achieving or beating the state-of-the-art performance is as important as understanding which are the limitations of those hyperparameters and testing the models under real-world conditions.

We have also studied the limitations resulting from environment non-stationarity. We have validated the Machine Learning community’s findings on the negative effects of non-stationary sources in the environment. It is not common to find this kind of analysis in domain-specific DRL papers. If the goal is make DRL deployable, it is essential that we address contingency cases in our problems, either by capturing the sources of non-stationarity in the training datasets, or by devising strategies to mitigate the impact during real-time operations. Otherwise we fail to meet the requirements of operating in real-world environments and, more importantly, reduce the reliability of our systems.

6. CONCLUSIONS

In this paper we have addressed the design and implementation tradeoffs of a Deep Reinforcement Learning (DRL) agent to carry out frequency assignment tasks in a multibeam
satellite constellation. DRL models are getting attention in the aerospace community mostly due to its fast decision-making and its adaptability to complex non-linear optimization problems. In our work we have chosen the Frequency Plan Design (FPD) problem as a use case and identified six elements that drive the performance of DRL models: the state representation, the action representation, the reward function, the policy, the policy optimization algorithm, and the training strategy. We have defined multiple variations for each of these elements and compared the performance differences in separate scenarios. We have put a special focus on high-dimensionality and non-stationarity, being two of the main phenomena present in the upcoming satellite communications landscape.

The results show that DRL is an adequate method to address the FPD problem, since it successfully assigns 85% to 99% of the beams for cases with 100 to 2,000 beams. However, no single state-action combination outperforms the rest for all cases. When the dimensionality of the problem is low, the Grid action space and the Without state representation perform better. In contrast, the Tetris-like action space and the With state representation are a better option for high-dimensional scenarios. These findings validate our hypothesis that representation should be strongly considered as an additional hyperparameter in applied DRL studies. We have also seen that using the Deep Q-Network algorithm in combination with a convolutional neural network and a fully-connected neural network as the policy works best for all scenarios, being especially advantageous for the 2,000-beam case. Regardless of the scenario, the obtained policy has shown a greedy behavior that benefits from informative rewards at each timestep.

At the end of the paper, we have reflected on different considerations that are usually left out of applied DRL studies. Our analyses on the effect of non-stationarity have helped motivating the discussion. When the average bandwidth demand per beam distribution substantially differs between the train and test data the number of successfully-assigned beams by agent decreases, performing worse than random for some cases. We emphasize the need to identify the potential sources of non-stationarity, understand its potential effects for the DRL model during real operation cycles, and propose solutions to mitigate any negative influences. A compromise between topping performance metrics and characterizing the limitation of the models and its hyperparameters is the only way to advance in the successful deployment of DRL and other Machine Learning-based technologies.

ACKNOWLEDGMENTS

This work was supported by SES. The authors would like to thank SES for their input to this paper and their financial support. The simulations are based on a representative example of a MEO constellation system, which was provided by SES [27]. The authors would also like to thank Ignacio del Portillo and Markus Guerster, for contributing to the initial discussions that motivated this work; and Skylar Eiskowitz and Nils Pachler, for reviewing the manuscript.

REFERENCES

[1] N. Abbas, Y. Nasser, and K. E. Ahmad, “Recent advances on artificial intelligence and learning techniques in cognitive radio networks,” Eurasip Journal on Wireless Communications and Networking, vol. 2015, no. 1, 2015.

[2] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, “Applications of Deep Reinforcement Learning in Communications and Networking: A Survey,” IEEE Communications Surveys & Tutorials, vol. 21, no. 4, pp. 3133–3174, 2019.

[3] P. V. R. Ferreira, R. Paffenroth, A. M. Wyglinski, T. M. Hackett, S. G. Bilen, R. C. Reinhart, and D. J. Mortensen, “Reinforcement Learning for Satellite Communications: From LEO to Deep Space Operations,” IEEE Communications Magazine, vol. 57, no. 5, pp. 70–75, may 2019.

[4] X. Hu, S. Liu, R. Chen, W. Wang, and C. Wang, “A Deep Reinforcement Learning-Based Framework for Dynamic Resource Allocation in Multibeam Satellite Systems,” IEEE Communications Letters, vol. 22, no. 8, pp. 1612–1615, 2018.

[5] X. Hu, Y. Zhang, X. Liao, Z. Liu, W. Wang, and F. M. Ghannouchi, “Dynamic Beam Hopping Method Based on Multi-Objective Deep Reinforcement Learning for Next Generation Satellite Broadband Systems,” IEEE Transactions on Broadcasting, vol. 66, no. 3, pp. 630–646, sep 2020.

[6] J. J. Garau Luis, M. Guerster, I. del Portillo, E. Crawley, and B. Cameron, “Deep Reinforcement Learning Architecture for Continuous Power Allocation in High Throughput Satellites,” arXiv preprint arXiv:1906.00571, 2019.

[7] P. Zhang, X. Wang, Z. Ma, S. Liu, and J. Song, “An online power allocation algorithm based on deep reinforcement learning in multibeam satellite systems,” International Journal of Satellite Communications and Networking, vol. 38, no. 5, pp. 450–461, sep 2020.
[8] Northern Sky Research, “VSAT and Broadband Satellite Markets,” Tech. Rep., 2019.
[9] Heng Wang, Aijun Liu, Xiaofei Pan, and Luliang Jia, “Optimal bandwidth allocation for multi-spot-beam satellite communication systems,” in Proceedings 2013 International Conference on Mechatron, Electric Engineering and Computer (MEC). IEEE, dec 2013, pp. 2794–2798.
[10] A. I. Aravanis, B. Shankar M. R., P.-D. Arapoglou, G. Danoy, P. G. Cottis, and B. Ottersten, “Power Allocation in Multibeam Satellite Systems: A Two-Stage Multi-Objective Optimization,” IEEE Transactions on Wireless Communications, vol. 14, no. 6, pp. 3171–3182, jun 2015.
[11] G. Cocco, T. De Cola, M. Angelone, Z. Katona, and A. I. Aravanis, B. Shankar M. R., P.-D. Arapoglou, G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester, “An empirical investigation of the challenges of real-world reinforcement learning,” mar 2020. [Online]. Available: http://arxiv.org/abs/2003.11881
[12] F. R. Durand and T. Abrisç, “Power allocation in multibeam satellites based on particle swarm optimization,” AEU - International Journal of Electronics and Communications, vol. 78, pp. 124–133, 2017.
[13] J. J. Garau Luis, N. Pachler, M. Guerster, I. del Portillo, E. Crawley, and B. Cameron, “Artificial Intelligence Algorithms for Power Allocation in High Throughput Satellites: A Comparison,” in 2020 IEEE Aerospace Conference. IEEE, mar 2020, pp. 1–15.
[14] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, “Deep reinforcement learning that matters,” in 32nd AAAI Conference on Artificial Intelligence, AAAI 2018, 2018.
[15] G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester, “An empirical investigation of the challenges of real-world reinforcement learning,” mar 2020. [Online]. Available: https://www.ses.com/networks/networks-and-platforms/o3b-mpower
[16] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press, 2018.
[17] I. Goodfellow, Y. Bengio, and A. Courville, Deep learning. MIT press, 2016.
[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, and Others, “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, pp. 529–533, 2015.
[19] M. Popova, O. Isayev, and A. Tropsha, “Deep reinforcement learning for de novo drug design,” Science Advances, vol. 4, no. 7, p. eaap7885, jul 2018.
[20] G. Theocharous, P. S. Thomas, and M. Ghavamzadeh, “Personalized ad recommendation systems for life-time value optimization with guarantees,” in IJCAI International Joint Conference on Artificial Intelligence, 2015.
[21] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Van Den Dien, T. Graepel, and D. Hassabis, “Mastering the game of Go without human knowledge,” Nature, 2017.
[22] N. Pachler de la Osa, M. Guerster, I. Portillo Barrios, E. Crawley, and B. Cameron, “Static beam placement and frequency plan algorithms for LEO constellations,” International Journal of Satellite Communications and Networking, p. sat.1345, aug 2020.
[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” arXiv preprint arXiv:1707.06347, 2017.
[24] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, “OpenAI Baselines,” https://github.com/openai/baselines, 2017.
[25] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” Neural Computation, 1997.
[26] J. J. Garau Luis, “A Comparison of Artificial Intelligence Algorithms for Dynamic Power Allocation in Flexible High Throughput Satellites,” Master’s thesis, Massachusetts Institute of Technology, 2020.
[27] SES, “SES mPower system.” [Online]. Available: https://www.ses.com/networks/networks-and-platforms/o3b-mpower

**BIography**

Juan Jose Garau Luis is a Ph.D. Candidate in the Department of Aeronautics and Astronautics at MIT. In 2017, under the CFIS program, he received two B.Sc. degrees in Telecommunications Engineering and Industrial Engineering from Universitat Politècnica de Catalunya, in Spain. In 2020, he received his M.Sc. degree in Aerospace Engineering from MIT. He has worked as a data scientist and Machine Learning engineer at Novartis, Arcvi, and the Barcelona Supercomputing Center. His research interests include autonomous systems, Machine Learning, and communication networks.

Prof. Edward Crawley received an Sc.D. in Aerospace Structures from MIT in 1981. His early research interests centered on structural dynamics, aeroelasticity, and the development of actively controlled and intelligent structures. Recently, Dr. Crawley’s research has focused on the domain of the architecture and design of complex systems. From 1996 to 2003 he served as the Department Head of Aeronautics and Astronautics at MIT, leading the strategic realignment of the department. Dr. Crawley is a Fellow of the AIAA and the Royal Aeronautical Society (UK), and is a member of three national academies of engineering. He is the author of numerous journal publications in the AIAA Journal, the ASME Journal, the Journal of Composite Materials, and Acta Astronautica. He received the NASA Public Service Medal. Recently, Prof Crawley was one of the ten members of the presidential committee led by Norman Augustine to study the future of human spaceflight in the US.
Dr. Bruce Cameron is a Lecturer in Engineering Systems at MIT and a consultant on platform strategies. At MIT, Dr. Cameron ran the MIT Commonality study, a 16 firm investigation of platforming returns. Dr. Cameron's current clients include Fortune 500 firms in high tech, aerospace, transportation, and consumer goods. Prior to MIT, Bruce worked as an engagement manager at a management consultancy and as a system engineer at MDA Space Systems, and has built hardware currently in orbit. Dr. Cameron received his undergraduate degree from the University of Toronto, and graduate degrees from MIT.