Deep Reinforcement Learning Applied in Elastic Optical Networks Using Game Representation

Moshe Simon, Rafael Asensio, Andrey Staniukynas,
Alina Stanoiu, David Sal

4th Year Project Final Report

Department of Electronic &
Electrical Engineering

UCL

Supervisor: Dr Alejandra Beghelli

26 April 2023

I have read and understood UCL's and the Department's statements and guidelines concerning plagiarism.

I declare that all material described in this report is my own work except where explicitly and individually indicated in the text. This includes ideas described in the text, figures and computer programs.

This report contains 50 pages (excluding this page and the appendices) and 13125 words.

Signed: _____ Date: _____

(Student)

Signed: _____ Date: _____

(Student)

Signed: _____ Date: _____

(Student)

Signed: _____ Date: _____

(Student)

Signed: _____ Date: _____

(Student)

Signed: _____ Date: _____

(Student)

Deep Reinforcement Learning Applied in Elastic Optical Networks Using Game Representation

Moshe Simon, Rafael Asensio, Andrey Staniukynas, Alina Stanoiu, David Sal

Executive summary
The rapid pace of digital transformation is driving network technologies to new heights, opening up the possibilities for more complex connected-systems applications. However, this progress poses a challenge for control systems to function effectively in highly dynamic network scenarios, with existing network optimisation technologies falling short of meeting their real-time demands.

Elastic Optical Networks (EONs) provide a potential solution to the increasing global network traffic by utilising bandwidth more efficiently. The aim of this project is to exploit these benefits by continuing previous research using Deep Reinforcement Learning (DeepRL) to solve routing and spectrum allocation (RSA) in EONs. Specifically, the project focuses on training a DeepRL agent on an arcade game representation of the RSA problem and evaluating its performance against KSP-FF, a state-of-the-art heuristic algorithm.

The contribution of this work is two-fold. Firstly, we optimise the pre-existing RSA game and add a novel approach of representing the routing issue visually. Multiple methods of optimising the game were explored such as redesigning the action space, experimenting with episode-ending mechanisms, and changing the number of frequency slot units (FSU) and pre-calculated paths. Moreover, in addition to the DQN algorithm used in the previous study, we also implemented the DreamerV8 agent. Secondly, besides the RSA optimisation attempts, a new game was developed focusing only on the routing problem, performing spectrum allocation using the first-fit (FF) algorithm.

Despite the promising results of the novel routing-only arcade game approach, which showed the best performance compared to other methods, it still fell short of the baseline heuristic by 10%. Although using certain setups the agent came close to the heuristic's performance and even outperformed it, overall it was unable to surpass the heuristic. Further analysis revealed that the main reason for this discrepancy was the lower episode lengths, which were approximately 40% shorter for the heuristic runs.

Overall, the findings indicate that breaking down the RSA problem into routing and spectrum allocation sub-problems leads to improved overall performance. Furthermore, it was demonstrated that simplifying the action space and providing the agent with immediate feedback after each action, facilitates more efficient learning.

Contents

1 Introduction **3**

2 Literature Review **4**
- 2.1 Heuristic approaches . . . 4
- 2.2 Reinforcement learning approaches . . . 5
  - 2.2.1 Numerical approaches solving routing . . . 5
  - 2.2.2 Numerical approaches solving RSA/RMSA . . . 5
  - 2.2.3 Visual representation solving RSA/RMSA . . . 6
  - 2.2.4 Summary . . . 7

3 Project Goals **7**

4 Technical Background of Your Work **8**
- 4.1 Optical Networks . . . 8
- 4.2 Elastic Optical Networks . . . 9
  - 4.2.1 Routing Algorithms . . . 11
  - 4.2.2 Modulation Formal Selection . . . 11
  - 4.2.3 Spectrum Allocation Algorithms . . . 12
- 4.3 Deep Reinforcement Learning . . . 12
  - 4.3.1 Reinforcement Learning . . . 12
  - 4.3.2 Deep Learning . . . 13
  - 4.3.3 Deep Reinforcement Learning . . . 14

5 Methods **14**
- 5.1 Upgrading the RSA Game . . . 15
  - 5.1.1 Game 6: Global Configuration . . . 15
  - 5.1.2 Game 7: Network Visualisation . . . 20
  - 5.1.3 Game 8: Combining Improvements . . . 24
  - 5.1.4 Improving DQN . . . 25
  - 5.1.5 Dreamer V3 World Model . . . 25
- 5.2 Routing Game . . . 26
  - 5.2.1 Environment Design . . . 26
  - 5.2.2 Model Implementation . . . 30
  - 5.2.3 Training and Evaluation . . . 31
  - 5.2.4 Policy Analysis . . . 33

6 Results and Discussion **33**
- 6.1 RSA Game . . . 33
  - 6.1.1 Game Optimisations . . . 33
  - 6.1.2 Game 7 . . . 37
  - 6.1.3 Game 8 . . . 38
  - 6.1.4 Dreamer . . . 39
- 6.2 Routing Game . . . 40
  - 6.2.1 Initial Results . . . 40
  - 6.2.2 Bug Fix . . . 41

6.2.3 Final Results 42
6.2.4 Analytics 44
6.3 Overall Results 46

7 Conclusions **46**

8 Team Contributions **47**

A Spectrum Allocation Data Structures **51**
A.1 Link Grid 51
A.2 Running Path Spectrum (RPS) 51

B Agent Game Implementation **52**

1 Introduction

With the growing digitalization of society, optical fibre networks have become essential for high-speed and high-bandwidth telecommunications globally. The Compound Annual Growth Rate (CAGR) for global internet traffic is projected to be 24% between 2021 and 2026 [1] which leads to a nearly 3-fold increase over the 5-year span. As a result, there is a need to improve the performance of optical networks to keep up with the increasing demand for traffic.

The prevalent optical fibre technology is currently fixed-WDM (Wavelength-Division Multiplexing), which uses a fixed grid to divide the available bandwidth into channels of 50 or 100 GHz [2], enabling transmission of multiple data signals on a single fiber. However, this technology has a rigid bandwidth allocation that assigns 50/100 GHz of bandwidth even if less is needed for small bandwidth signals, resulting in unused spectrum. Furthermore, the fixed grid used in fixed-WDM can limit the transmission of signals requiring more than 50/100 GHz of spectral bandwidth, which can restrict the potential applications of fixed-WDM as higher bandwidths are increasingly needed for new applications like 5G and IoT. For instance, transmitting uncompressed 8K video may require up to 144 GHz of bandwidth, which cannot be accommodated by the fixed grid, further underscoring its limitations [3].

To overcome the shortcomings of fixed-WDM researchers have been exploring Elastic Optical Networks (EONs) as a potential solution [4]. The flexible grid approach of EONs divides the spectrum into smaller 12.5 GHz channels, with the option to allocate multiple channels for one signal. This provides more flexible spectrum allocation based on the bandwidth demand, meaning that smaller requests can be assigned a single small spectrum slot, while larger requests can be allocated multiple consecutive spectrum slots. However, the technology still faces various challenges, such as the need for hardware devices with better performance, effective network control plane management, resource allocation and intelligent routing, and post-disaster management and recovery [5].

This study addresses the RSA (Routing and Spectrum Assignment) problem, which consists of finding a feasible route and spectrum allocation for a given set of traffic demands while satisfying the contiguity and continuity constraints. Specifically, EONs require that spectrum slots allocated on one link for one request must be next to each other

(contiguity), and the same slots must be allocated on all links from the source to the destination (continuity). These requirements will be discussed in more depth in the Technical Background section. To extend RSA, the RMSA (Routing, Modulation, and Spectrum Assignment) problem considers the modulation format that can be used on each link to maximize spectral efficiency.

Using to solve the resource allocation problem is a significant step towards the deployment of EONs. If successful, it could reduce the blocking probability and increase the network throughput, resulting in faster and more reliable communication within the telecommunications industry.

This paper is structured as follows. In the next section, an extensive review of the existing literature is presented, providing valuable insights into the field. The goals of the project are then outlined. This is followed by a background theory section that introduces the relevant concepts and theories. The methods section describes the implementation of this work, while the results section presents the experimental findings. Finally, the conclusions drawn from the results are summarized, and potential directions for future research are proposed.

2 Literature Review

There are multiple research papers that propose different approaches to solving the RSA and RMSA problems. These approaches can be divided into the following main categories: Integer Linear Programming (ILP), heuristic and machine learning approaches. ILP algorithms convert the RSA challenge into a mathematical optimization problem. Although these algorithms can provide an optimal solution, they are computationally intensive and their time complexity grows exponentially [6], making them unsuitable for solving larger problems. As a result, ILP algorithms are generally limited to smaller, simplified problems.

2.1 Heuristic approaches

There are numerous heuristic approaches, with all of them exhibiting strengths in different domains. K-shortest-route routing with first-fit spectrum assignment (KSP-FF) is one of the commonly used approaches that generally perform well and have low computational complexity [7]. This algorithm is usually used as a benchmark in RSA/RMSA studies.

An example paper attempting the challenge using heuristic methods is presented in [8]. The paper proposed heuristic methods that consider fragmentation and congestion avoidance to achieve efficient spectrum allocation. While results outperform both benchmark algorithms in a 14-node network the proposed heuristic algorithms failed to exceed the performance of the benchmarks in a larger 28-node network. Another paper [9], proposed a heuristic that finds the common longest unused spectrum segment between two adjacent links and allocates it to the corresponding traffic demands, aiming to minimize fragmentation and increase spectrum utilization. This approach achieved similar results.

A detailed analysis and comparison of all the available heuristic algorithms for RSA/ RMSA in EONs can be found in [10]. The analysis revealed that heuristic algorithms exhibit varying strengths in different areas such as blocking probability, cost, effective resource management, quality of spectrum provisioning, and more. However, no single algorithm can be considered as an ultimate solution to the RSA/RMSA problem that would outperform other methods in all areas.

2.2 Reinforcement learning approaches

Heuristics are algorithms that efficiently approximate the optimal solution, relying on the network state and topology. As the size of the topology is increased, the algorithms can face scalability issues or provide sub-optimal solutions. To address this challenge, researchers are exploring Reinforcement Learning (RL) as a potential alternative, with the goal of achieving a more scalable and autonomous solution to the resource allocation problem in EONs. This category can be further classified into two subcategories based on how the network is represented: numerically or visually with the latter being a new area of research. Another way to categorize approaches is based on the specific problem being addressed, which can be further divided into routing only and routing with spectrum allocation (RSA).

2.2.1 Numerical approaches solving routing

Many researchers attempted to simplify the resource allocation problem by separating routing from spectrum allocation. In [11–13] the reinforcement learning agent is only trained to perform routing for the requests while for spectrum allocation a heuristic was used. [12] uses PPO (Proximal Policy Optimization) as their agent while [11, 13] apply a network-specific TRP (Traffic and Resource Provisioning) agent.

Suárez-Varela et al. [11] identifies that previous works have a limitation where the agent's observation space contains only link-level features, while the action space is defined at the route level. The agent needs to abstract information between different levels due to this mismatch. Additionally, previous representations do not account for network topology and interdependencies between links forming end-to-end routes. In the proposed representation, the state space is depicted as a matrix that contains statistics for k end-to-end routes across all source-destination combinations. This new approach outperforms existing state-of-the-art methods in two different traffic scenarios while achieving comparable results to current heuristics. Opposed to [12], this paper develops a new routing algorithm through reverse engineering the agent's policy, which demonstrates superior performance compared to SAP (Shortest Available Path) baseline heuristic.

In contrast to [11], Almasan et al. [12] propose to represent the network using Graph Neural Networks (GNN). The observation space includes the utilization of current links in the network, which is passed through a GNN to learn a graph representation. At the beginning of each episode, routing requests are allocated using the Open Shortest Path First (OSPF) algorithm, and then the DeepRL agent iterates through each element of the graph to optimize the routing policy and minimize the utilization of the most loaded link. The results show that the DeepRL agent outperforms the baseline heuristics, SAP and Simulated Annealing, in terms of routing policy and computational time. The agent outperforms Simulated Annealing in terms of optimization time, taking only 0.15s compared to the latter's 1,878s, making it a promising solution for practical applications.

2.2.2 Numerical approaches solving RSA/RMSA

Chen et al. [14] address the RMSA problem in EONs which also takes the modulation format into account besides attempting the RSA challenge. The authors propose a numerical state space using a binary matrix that indicates the availability (0-occupied, 1-available) of frequency slots on the links. The resultant model showed superior performance compared to KSP-FF, with a 20.3% reduction in blocking probability, which refers to the probability of a request getting rejected. It was pointed out that conducting training

sessions with a large number of connection requests was computationally expensive and suggested room for improvement in this aspect.

In [15] Chen et al. provide a comprehensive description of the automatic framework used for implementing the DeepRMSA agent, in contrast to their previous paper [14] where the focus is on detailing the DeepRL agent. Their analysis highlights the advantages of DeepRMSA over baseline heuristics and extends the framework to accommodate multi-domain EONs. The performance of the DeepRL agent is compared with two baseline heuristics showing a reduction in blocking probability by 23.9%.

Nevin et al [18] discusses a numerical approach by applying RL with invalid action masking and a new training methodology for RSA in fixed-grid optical networks. The authors incorporated domain knowledge in the RL model by constraining the action space to routes that can support the current request. The model was trained on a simplified problem and outperformed standard RL and three state-of-the-art heuristics on the target RSA problem. The RL agent trained on uniform traffic generalized well to nonuniform traffic, outperforming the heuristics. Analysis of the learned policy revealed a different strategy compared to the heuristic baselines in service distribution across channels and links.

In [19], the importance of proper reward design in DeepRL framework is highlighted by Tang et al. Similarly to [18] they propose incorporating heuristic information to reduce exploration blindness and improve learning efficiency. The authors use the spectrum fragmentation level of each candidate route as the additional information to guide the DeepRL agent towards selecting routes with lower fragmentation levels that are more suitable for future traffic requests. Results suggested a superior performance of the proposed heuristic reward design scheme when compared with Standard-DeepRL and heuristic approaches. Nevertheless, the authors highlight that the issue of determining what type of domain knowledge should be leveraged into the reward design remains unresolved.

2.2.3 Visual representation solving RSA/RMSA

Ziazet [20] addresses the RMSA problem and trains three DeepRL agents, DQN, REINFORCE and A2C in contrast to [21,22] who only explored DQN. The approach calculated k shortest routes, modulation format and the number of slots per request, and passed a visual representation of the network displaying the available links to a DeepRL agent. Results indicated that the REINFORCE algorithm was the most effective, but failed to outperform the baseline heuristic.

Ray et al. [21] approached the RSA problem by representing it as an arcade game interface. The game displayed the topology, connection requests, and spectrum of each link, and users selected nodes and spectrum slots to establish a route from the source to the destination node. Three DQN-based RL agents were trained on different traffic loads and evaluated on metrics such as blocking probability, cause of blocking, and average route length. The agents performed well in the arcade game but failed to outperform the baseline heuristic, KSP-FF, and could not generalize to untrained requests.

In a similar manner to [21], Moshe Simon [22] applied the DQL algorithm to a custom-made arcade game environment, but with a different interface. The agent was presented with the current request and pre-calculated k shortest routes where it could navigate through routes and spectrum slots using the LEFT and RIGHT arrow keys, and allocate spectrum by pressing ENTER. Although the results indicated that the top-performing

model, named Silver-glade-5, was unable to outperform the KSP-FF heuristic used as a benchmark, the study demonstrated the potential of visual representations.

2.2.4 Summary

An overview of the explored literature is presented in Figure 1. The literature review has shown the widely recognized potential of RL as a solution to the RSA problem, but also certain limitations within the approaches. Proposed methods are often complex and computationally intensive, making the training process difficult and time-consuming. Additionally, the models may be challenging to interpret, and some methods have inconsistent or worse performance when compared with existing heuristic techniques.

RefNr	Published	Title of the research paper	Link	Problem	Agent used	Representation	Baseline heuristic	Performance metric	Best performance	Reverse Engineer
[11]	24/09/2019	Routing in optical transport networks with deep reinforcement learning	link	Routing	TRPO	Numerical	SAP	Bw allocated	RL better overall but not consistent	Yes
[12]	17/06/2021	Towards Real-Time Routing Optimization with Deep Reinforcement Learning: Open Challenges	link	Routing	PPO	Numerical	SAP, Simulated Annealing	Maximum link utilization	RL wins by 15% and 4% for two different topologies	No
[13]	22/09/2022	Routing and Spectrum Assignment Assisted by Reinforcement Learning in Multi-band Optical Networks	link	Routing	TRPO	Numerical	K-SP-HighestGSNR, K-SP-LowestGSNR, K-SP-FF	Blocking probability	RL beats heuristics but for high load only slightly	No
[14]	15/08/2019	DeepRMSA: A Deep Reinforcement Learning Framework for Routing, Modulation and Spectrum Assignment in Elastic Optical Networks	link	RMSA	A3C	Numerical	SP-FF, KSP-FF	Blocking probability	RL beats by 20.3%	No
[15]	17/10/2019	Building Autonomic Elastic Optical Networks with Deep Reinforcement Learning	link	RMSA	A3C	Numerical	SP-FF, KSP-FF	Blocking probability	RL beats by 23.9%	No
[16]	06/12/2020	A Reinforcement Learning Framework for Parameter Optimization in Elastic Optical Networks	link	Similar to RSA	A2A	Numerical	GA	Bitrate	Comparable, RL higher variance	No
[17]	05/07/2021	Resource Allocation in Multicore Elastic Optical Networks: A Deep Reinforcement Learning Approach	link	RMSCA	TRPO, PPO2, A2C, ACKTR	Numerical	kSP-FF-FCA, KSP-RF-RCA, KSPSCMA XT/demand Aware	Blocking probability	Low traffic: RL High traffic: similar	No
[18]	09/09/2022	Techniques for applying reinforcement learning to routing and wavelength assignment problems in optical fiber communication networks	link	RWA	PPO	Numerical	kSP-FF, kSP-MU, FF-kSP	Services accepted	Standard RL fails but invalid masked RL beats heuristics	Yes
[19]	25/11/2022	Heuristic Reward Design for Deep Reinforcement Learning-Based Routing, Modulation and Spectrum Assignment of Elastic Optical Networks	link	RMSA	A3C	Numerical	KSP-FF, KMC-FF, KMF-FF	Blocking probability	Heuristic	No
[20]	20/12/2019	Reinforcement Learning for Routing, Modulation And Spectrum Assignment Problem in Elastic Optical Networks	link	RMSA	DQN, A2C, REINFORCE	Visual	Adapted Compact Scheduling Algorithm	Total throughput	REINFORCE performs best and comes close, but doesn't beat Heuristics	No
[21]	unpublished 2022	Artificial Intelligence in Elastic Optical Networks	AB	RSA	DQN	Visual	KSP-FF	Game score	Heuristic	No
[22]	unpublished 2022	Deep reinforcement learning applied in dynamic elastic optical networks using arcade-like game representation	AB	RSA	DQN	Visual	KSP-FF	Blocking probability, Avg routelength	RL higher avg score but does not beat heuristic consistently	No

Figure 1: EON Literature Review Summary Table

This work proposes to represent the problem visually as an arcade game, building upon the novel approach proposed by Moshe Simon [22] with the aim of improving performance by leveraging successful methods discovered during the literature review. Furthermore, the present study proposes a novel approach addressing the routing issue only using a visual approach, inspired by the noteworthy results on numerical representation approaches from the literature review [11–13]. The hope is that utilising these methods and simplifying the problem will result in increased performance of the agent, while the visual representation will provide intuitive interpretation.

3 Project Goals

1. Set up an experimentation framework.
   (a) Produce a global configuration file to manage all the parameters used by different files for a game.
   (b) Generate a naming convention to keep track of parameter tuning and changes to the game.
2. Boost RSA agent performance.

(a) Experiment with non-structural changes to the game to increase performance.

(b) Test network visualisation features for the agent in-game.

(c) Investigate the latest advancements in DeepRL and possible implementation in the game (including world models).

3. Create a Routing-only game.

   (a) Structure GUI for the agent to interact with the game effectively.

   (b) Set up spectrum allocation heuristics so that the agent can focus on routing purely.

4. Implement an agent analytics platform.

   (a) Gather episode analytics in both heuristics and agent runs to compare and draw insights into drivers of performance.

4 Technical Background of Your Work

present and explain the mathematics, science, and technology that forms the basis of your project in your own words.

4.1 Optical Networks

An optical network is a type of computer network that relies on optical fibers to transmit information. In such networks, data is typically encoded onto a beam of light using various modulation formats and then propagated through optical fibers to the intended destination. These networks are widely used in telecommunications, data centers, and cloud computing to support the ever-increasing demand for high-speed data transmission.

One of the key technologies used in optical networks is wavelength-division multiplexing (WDM), which allows the transmission of multiple wavelengths at the same time through the optical fiber by multiplexing the modulated signals. When the multiplexed signal is transmitted over the fibre this will be both amplified and switched in the optical domain. Only at the receiver end the signal will be de-multiplexed and converted back to the electrical domain [23].

The traffic in an optical network can be static or dynamic. For both types, a request or demand contains a source, a destination and bandwidth. In a static optical network, the traffic demand is pre-planned and the network resources are allocated accordingly and remain fixed over time. This type of traffic is most commonly used today since it is simple and cost-effective. However, for static traffic, the allocated resources remain reserved at all times, even if they are not utilized constantly. In dynamic optical networks, the resources are allocated to a given connection only when these are needed. In this case, the traffic demands are not pre-planned but arrive in the network as a sequence of events and resources have to be allocated fast and for a limited amount of time based on the network state at that time. Figure 2 presents a comparison between the two types of traffic in optical networks. As can be seen, dynamic traffic only requires one wavelength to allocate resources whereas, static traffic requires two.

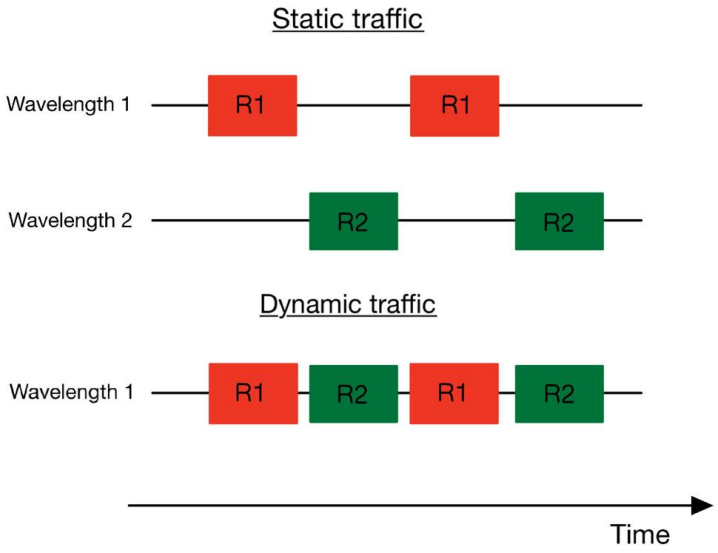


Figure 2: Static and Dynamic traffic in Optical networks

In the case of dynamic optical networks, the traffic is described using the following concepts: inter-arrival time and holding time. The inter-arrival time describes the time that elapses between two consecutive request arrivals. The holding time describes the time that elapses between the arrival and departure of a certain connection from the network. These two concepts are represented in Figure 3. Therefore, the traffic load of an dynamic optical network can be calculated as the ratio between the mean holding time and the mean inter-arrival time.

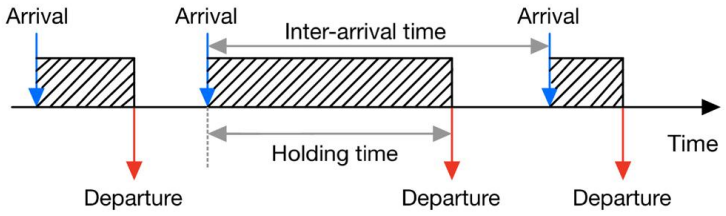


Figure 3: Dynamic Traffic

4.2 Elastic Optical Networks

In fixed-grid optical networks, most commonly used in networks today, the spectrum is divided into a set of predefined wavelengths or frequency channels used for wavelength-division multiplexing (WDM) transmission. These channels are typically separated by a fixed frequency interval which is called guard band with a bandwidth of 50 or 100 GHZ depending on the band [2]. Fixed-grid networks offer standardization and cost-effective use of the spectrum, but they also have limited flexibility which leads to a lot of wasted spectrum. This is represented in the upper part of Figure 4. Nowadays, data in optical networks is transmitted at various bit rates and requires different amounts of bandwidth. Using the fixed grid optical network for this type of data and especially for dynamic traffic results is an inefficient use of the spectrum. For example in a network with 50 GHz spacing,

10 and 40 Gbps signals do not use the allocated bandwidth efficiently, while signals that have a bit rate higher than 100 Gbps cannot be supported by the network. The idea of elastic optical networks(EON) is to give each connection the spectral width that it needs. EON offers a flexible grid by dividing the spectrum into 12.5 GHz portions called Frequency Slot Units (FSU). Depending on the spectrum needed by each connection, one or multiple FSUs will be allocated to each connection, so each signal receives multiples of 12.5 GHz [23, 24] as depicted on the lower side of Figure 4.

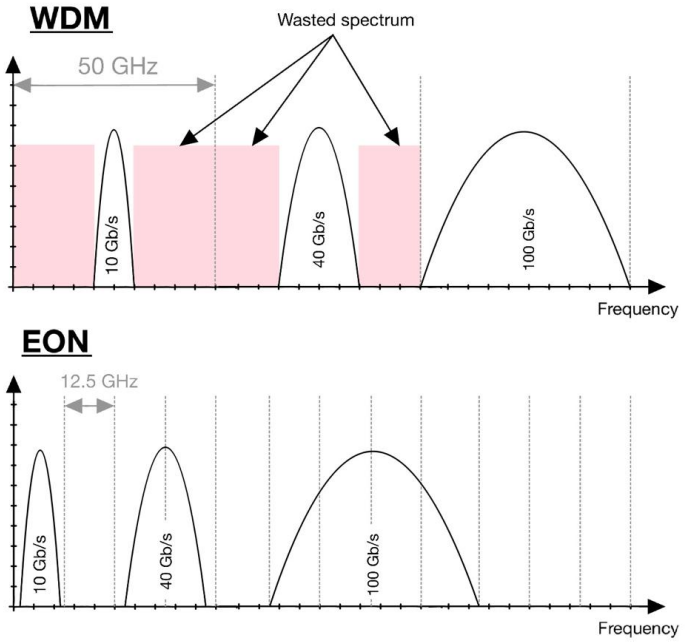


Figure 4: Fixed-grip Optical networks vs Elastic Optical Networks

In the context of EONs, the provisioning of resources is referred to as Routing and Spectrum Allocation (RSA) problem. Given a connection request (source, destination, bit rate) a route and a set of FSUs in that route have to be allocated for the connection to be established. The set of FSUs that are allocated has to comply with two constraints: continuity and contiguity, presented in Figure 5 and 6. The continuity constraint requires that exactly the same set of slots are used along the entire route [23, 24].

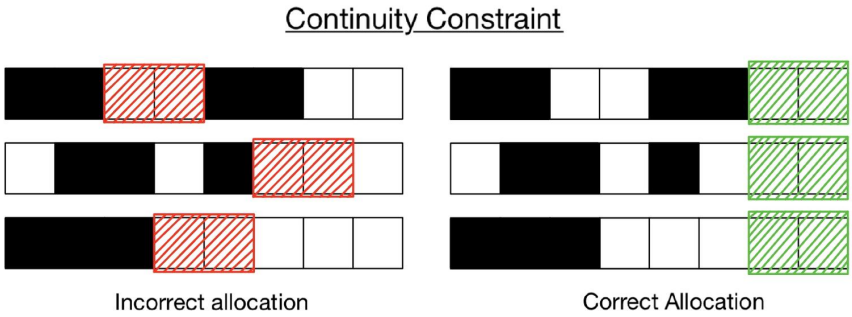


Figure 5: Continuity Constraint

For the contiguity constraint, the slots must be next to each other in the spectrum of each link.

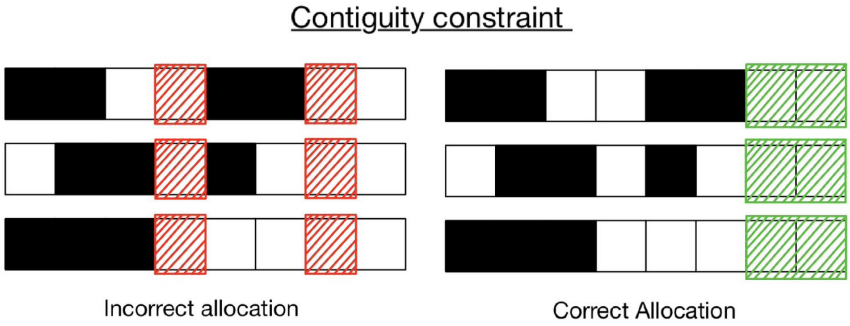


Figure 6: Contiguity Constraint

Usually, the request comes in bit rates but the spectrum is allocated in FSUs. The number of slots required for a signal with a given bit rate depends on the modulation format used. If the modulation format is already given or fixed, then only the RSA problem has to be solved. Otherwise, the provisioning problem becomes the Route, Modulation and Spectrum Allocation (RMSA) problem. In this case, a new trade-off between the optical reach and the number of slots is introduced. Some modulation formats require very few slots but also have a very short optical reach [23, 24].

One way of solving the resource allocation problems mentioned above is using heuristic algorithms, which solve each problem independently.

4.2.1 Routing Algorithms

There are three main routing algorithms for determining a route in a network: fixed routing, alternate routing and adaptive (dynamic) routing. The fixed routing algorithm receives a table with each source-destination pair and pre-computes route for each pair in the table using a shortest-route algorithm such as Dijkstra. When a connection request is received, the algorithm calculates the number of slots required for that connection and checks whether they are available on all links in the pre-computed route. If any slot is unavailable, the connection request is denied [23, 24].

The alternate routing algorithm works similarly with fixed routing but pre-computes an ordered list of fixed routes for each pair in the table. The same algorithm as before is used to compute the shortest routes. When a connection request is received, the algorithm checks if the spectrum in the first route of the table is available. If the necessary slots are not available, the algorithm moves on to the next route in the list and repeats the process.

In adaptive routing, no pre-computed routes are available and a route is computed based on the current network state at the time of the request arrival. Although this method leads to a significant decrease in blocking probability, it also entails a considerable increase in computation time.

4.2.2 Modulation Formal Selection

In elastic optical networks, the modulation format used for transmitting data signals is typically selected after a route has been established. Selecting the modulation format

requires minimising the number of frequency slots used (FSUs) and ensuring the signal can travel the required distance (optical reach).

4.2.3 Spectrum Allocation Algorithms

Once the route and if required the modulation format have been selected, a spectrum allocation algorithm is used to choose the FSUs. Several options exist for spectrum allocation algorithms, including first-fit, random-fit, min-max-fit, and last-fit. Only the most commonly used ones will be discussed in this section.

In First-Fit spectrum allocation algorithm an indexed list of available and used slots is maintained. The algorithm allocates the spectrum from the lowest to the highest index slot in order. This approach ensures that more spectrum slots remain available for other connection requests, thereby resulting in a low blocking probability and low computational complexity.

The Random Fit algorithm maintains a similar list but selects a random slot for allocation to a new route. This approach decreases the likelihood of multiple connections choosing the same slot. However, among spectrum allocation policies with low time complexity, Random Fit has the highest blocking probability, which represents its main drawback [24].

4.3 Deep Reinforcement Learning

4.3.1 Reinforcement Learning

Reinforcement learning (RL) is a branch of machine learning that aims to train agents to make decisions based on their interactions with an environment. These agents seek to maximise the accumulated reward over time through trial and error, adjusting their actions in response to the feedback they receive. RL has been successfully applied to a wide range of applications, including robotics, natural language processing and arcade games.

Markov Decision Processes

One common framework used to model reinforcement learning problems is the Markov Decision Process (MDP). MDPs consist of a tuple (S, A, P, R, γ), where S represents the set of states, A represents the set of actions, P represents the state transition probabilities, R represents the reward function and γ is the discount factor. The policy, π, is a function that maps states to actions and is represented as follows:

$$\pi = P(\alpha \mid s) \tag{1}$$

The goal of reinforcement learning is to find an optimal policy π^* that maximises the expected return R_t, which is the discounted sum of future rewards:

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \tag{2}$$

MDPs are particularly suitable for RL problems because they have the Markov property, which states that the future dynamics of the environment depend only on the current state and not on the history of past states.

Q-Functions

A Q-function, or action-value function, represents the expected return for a given state-action pair when following a policy π. The Q-function for a policy π is denoted as $Q_\pi(s, a)$ and is defined as:

$$Q_\pi(s, a) = E_\pi \lfloor R_t \mid S_t = s, A_t = a \rfloor \tag{3}$$

The optimal Q-function, $Q^*(s, a)$, is the maximum Q-function over all possible policies:

$$Q^*(s, a) = \max_\pi Q_\pi(s, a) \tag{4}$$

Q-functions are essential in RL as they allow agents to estimate the value of different actions and select the ones that maximize their expected return.

Bellman Equation

The Bellman equation is a recursive relationship that defines the optimal Q-function in terms of the expected immediate reward and the discounted future value of the optimal action in the next state:

$$Q^\star(s, a) = E_{s'} \left[r + \gamma \max_{a'} Q^\star \left(s', a' \right) \mid s, a \right] \tag{5}$$

The Bellman equation serves as the foundation for many RL algorithms, as it allows for the iterative estimation and refinement of Q-function values. The loss function for learning the Q-function can be defined as the squared difference between the current Q-value estimate and the optimal Q-value:

$$\left(Q^*(s, a) - Q(s, a) \right)^2 = \text{ loss} \tag{6}$$

One common method for learning the Q-function is Q-learning, which iteratively updates the Q-function estimates using a learning rate α and the following update rule:

$$Q_{i+1}(s, a) = (1 - \alpha) Q_i(s, a) + \alpha \left(E_{s'} \left[r + \gamma \max a' Qi \left(s', a' \right) \mid s, a \right] \right) \tag{7}$$

Q-learning is a model-free, value-based method that converges to the optimal Q-function under certain conditions, such as sufficient exploration and appropriate learning rate.

4.3.2 Deep Learning

Deep Learning (DL) is a sub-field of machine learning that focuses on neural networks with multiple layers, known as deep neural networks. These networks can learn representations of data with multiple levels of abstraction, making them effective at solving complex tasks such as image recognition, natural language processing, and more. Deep learning has been particularly successful in domains where there is a large amount of raw data available, such as images, audio, and text. Deep neural networks consist of multiple layers of interconnected neurons, which are organised into input, hidden, and output layers. The hidden layers enable the network to learn complex, non-linear relationships between the input and output data. The learning process involves training deep neural networks by adjusting the weights of the connections between neurons using gradient descent optimization and back-propagation algorithms.

4.3.3 Deep Reinforcement Learning

Deep reinforcement learning (DeepRL) combines reinforcement learning with deep learning techniques to learn policies and Q-functions using neural networks. In DeepRL, the Q-function is parameterised by a set of weights θ_i. The loss function for DeepRL is similar to the one for Q-learning but with the Q-function represented by a neural network:

$$\left(r + \gamma \max a' Q\left(s', a'; \theta_i\right) - Q\left(s, a; \theta_i\right)\right)^2 = \text{ loss} \tag{8}$$

The use of deep neural networks allows DeepRL algorithms to scale to problems with high-dimensional state and action spaces, making them suitable for a wide range of complex tasks.

Deep Q-Learning

Deep Q-learning (DQN) is a popular DeepRL algorithm that uses a deep neural network to approximate the Q-function. DQN extends the Q-learning algorithm by using a neural network to represent the Q-function and learn the optimal policy. To stabilize learning, DQN employs two main techniques: experience replay and target network updates. Experience replay stores a buffer of past transitions and samples mini-batches of experiences for learning, breaking the correlation between consecutive samples. Target network updates involve periodically updating a separate target network to prevent the Q-function from chasing a moving target. These techniques have been crucial in enabling DQN to achieve state-of-the-art performance in various domains, such as Atari games and robotic control tasks.

World Models

World models are a class of DeepRL methods that use neural networks to learn an internal model of the environment. The internal model is typically used to simulate trajectories, which can then be used for planning and learning purposes. World models can be learned using unsupervised or self-supervised learning techniques, such as autoencoders or variational autoencoders (VAEs), which learn compact representations of the environment's state. They are beneficial for RL tasks because they enable agents to learn a rich perception of the environment and imagine potential future outcomes of actions.

In summary, deep reinforcement learning combines reinforcement learning with deep learning techniques to enable agents to learn complex policies and value functions in high-dimensional state and action spaces. Various DeepRL algorithms, such as Deep Q-learning and world models, have demonstrated significant success in the world of arcade games and have the potential to show promising results when applied to the RSA problem in an arcade game format.

5 Methods

The work was divided into two: experimenting with the pre-existing RSA game [22] to improve performance and our understanding of the agent's actions, and developing a new routing-only game to boost performance versus baseline heuristics.

To begin, it is necessary to specify the details of the EON simulated in this research. The network topology utilized consists of 6 nodes, shown in Figure 7. In optical networks, bi-directional links are used since every link is made of two fibres (one in each

direction), hence the data can flow in both directions simultaneously. In order to simplify the problem, it is assumed that once a slot is occupied, it is occupied in both directions (half-duplex links). A simplified case of dynamic resource allocation is adopted - only one request is allocated at a time, and a new request arrives only after the first one has been allocated. This emulates a centrally controlled system with a single processing unit. Furthermore, the holding time of a request is considered to be infinite once the resource has been allocated. To solve the RSA problem, continuity and contiguity must be followed.

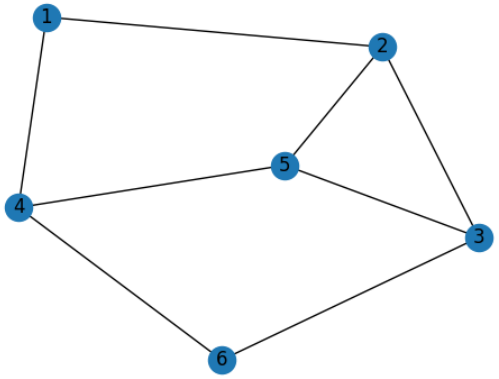


Figure 7: Network Topology

5.1 Upgrading the RSA Game

Building upon the work done by M. Simon [22], where Games 1-5 were developed (more on these games outlined in the paper), we developed Games 6-8 incorporating major improvements and design changes. Every game has two versions: one for human players and one for DeepRL agent, which are effectively identical (except in how the player interacts with the game). The human-playable version is vital in order to test the game, and understand it's dynamics and internal systems.

5.1.1 Game 6: Global Configuration

The following improvements were introduced with respect to the baseline game:

- Central configuration file.
- New naming convention for runs and saved files.
- New environment.
- New game-ending mechanism.
- Variable game length.

Central Configuration File

A central configuration file was motivated by a large number of reused variables and functions across project files. As we made changes to the agent's configuration, keeping track of the experiments was becoming increasingly difficult with each layer of complexity added to the project. Thus, the parameters along with reusable functions, were put in

one file in order to keep them consistent, allowing us to organise our experiments more efficiently. The configuration file includes the following parameters:

- **Game parameters**: game number, information about game window, colour definitions.
- **Environment parameters**: environment number, solution reward, rejection reward, environment type, game end mechanism type, number of slots, number of routes K.
- **Agent hyperparameters**: batch size, buffer size, epsilon, exploration fraction, gamma, learning rate, learning starts, target update interval, training frequency.
- **Training parameters**: total time-steps, seed.
- **Evaluation parameters**: number of episodes evaluated, evaluation seed.

Moreover, a new file naming convention was created to reflect model parameters, making comparison of results and plotting significantly more straightforward, allowing us to better measure the impact of changes made.

Environments

- *Environment 1:* Arrows

  This was the original environment, where in order to select the slot a selection window needed to be moved left and right multiple times before confirming the selection. For the human-representation, this was done using the keys LEFT and RIGHT to move the selection window, and ENTER to confirm it. In the agent-representation, agent had three actions, only receiving a reward for selecting a slot (the equivalent of pressing ENTER). This meant that a reward is given after a sequence of actions, rather than a single action, which made it challenging to map an action to a reward.

- *Environment 2:* Teleport

  To make it easier to map a reward to an action, a different gym environment was created. Instead of having to move a selection window until the desired position is set, the player can input number of a desired position as the action directly. If the width of the selection window is larger than 1 (which was the case across all experiments in this paper), the number represents position of the leftmost side of selection window. This required two additional checks:

  1. End of the selection window is not out of bounds: this would be the case if, for instance, the last route ends on slot 24, the selection window width is more than 2, and number 24 is picked as an action.
  2. Selection window is not split between two routes: this could happen if, for instance, the first route ends on slot 7, the second begins on slot 8, the selection window width is more than 2, and number 7 is picked as an action.

Size of the action space substantially increased and became equal to the number of possible slot locations on the screen. If agent selected one of the forbidden actions, it would result in the solution being ignored and agent receiving a `gap_reward`.This improvement resulted in the agent receiving a reward right after making an action, resulting in instantaneous feedback.

Episode Ending Mechanisms

- *Mechanism 1:* Failed requests

  Previously, a game would finish whenever a number the failed requests (those which were not allocated successfully) went over the threshold, controlled by the `end_limit` variable. These failed requests did not need to happen consecutively, but could be spread over the course of an episode.

  This posed two problems. Firstly, even if the agent was successful at filling out the entire grid and there was no spectrum left available, a certain number of requests needed to be allocated incorrectly in order to finish the episode. This meant that the agent was penalised even if it behaved perfectly throughout the episode. Secondly, if the number of maximum allowed failed requests differed between two trainings, it was difficult to compare and evaluate the two. Therefore, when comparing the results, we adjusted the reward of every episode by adding $end\ limit \times rejection\ reward$, given that the agent would be penalised by this amount in each episode.

- *Mechanism 2:* Total requests

  In order to tackle those issues, we introduced a new episode-ending mechanism. Now, the metric used to determine if an episode was finished was overall requests, instead of failed requests. As a result, the episode would end only when a certain number of requests was allocated (set by `end_limit`), regardless of whether or not they were successful. This solved the problem of forcefully penalising the agent at the end of an episode, but posed new issues.

  It is crucial to set the number of allowed requests per episode high enough so that nearly all slots are filled, but not so high that the agent has no allocation options for multiple rounds in a row. If the number was too high, it would result in the same problem as with Mechanism 1: the agent would encounter an entirely filled grid and would have to keep getting penalised until reaching the request limit. On the flip side, if the number was too low there could still be empty slots left on the grid when the episode was finished, risking under-fitting from not encountering high spectrum utilisation scenarios.

  This number was set experimentally based on our best estimate, with a different value for scenarios with different number of slots. It was calculated using the probability distribution of requested slot width (width of a selection window), as well as the number of slots and routes.

  For instance, if the width of a requested slot varied from 2 to 5 (meaning a 3.5 average) and there were 8 slots and 8 routes, `end_limit` would be set to 18 ($8 \times 8 \div 3.5 = 18.28$, rounded down). If there were 16 slots and 8 routes, `end_limit`

would be set to 36 ($16 \times 8 \div 3.5 = 36.57$, rounded down). If there were 24 slots and 8 routes, `end_limit` would be set to 54 ($16 \times 8 \div 3.5 = 54.85$, rounded down). In general:

$$end\ limit = slots \times routes \div average\ width$$

The values were picked with the aim that on average, an episode would finish as soon as the grid was full (in the case of a perfectly-acting agent).

Number of Routes and Slots

Another issue we looked at was having a variable number of slots and routes displayed. The numbers of slots available tested were 8, 16 or 24, and the numbers of routes (set through variable `K`) displayed to the agent were 1, 3 and 5. This meant that by setting `K` to 1 the routing problem was effectively eliminated, leaving spectrum allocation only. By trying out different combinations of these parameters along with combinations of the environment, episode ending mechanism and end_limit number, we were able to observe how they affected the performance.

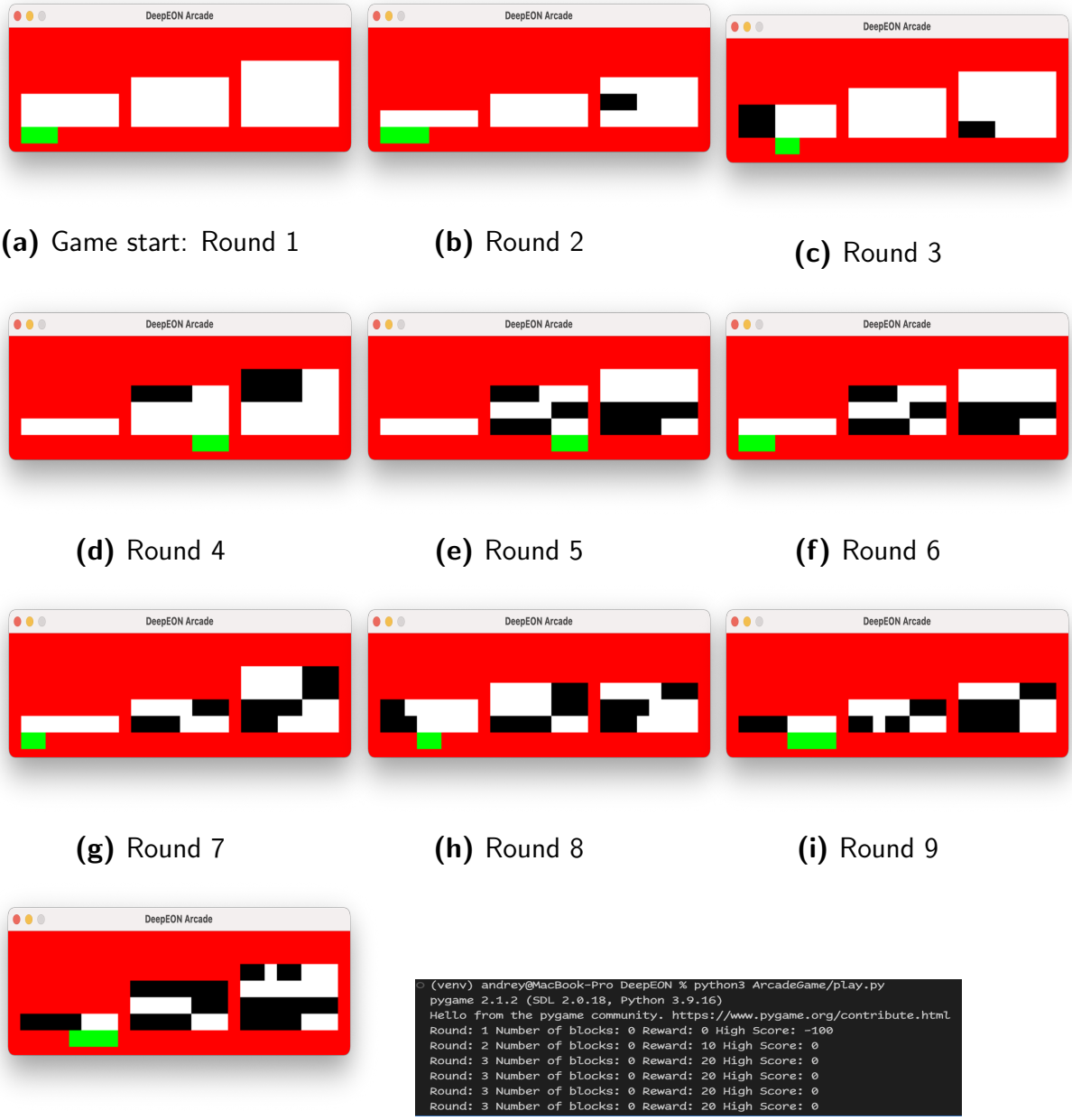
Parameter	Value
number of slots	8
number of routes displayed	3
environment	1
episode end	1
end limit	1
request slot width	[2, 3, 4, 5]
solution reward	10
rejection reward	-10
rejection reward	-5

Table 1: Game 6 example parameters.

Figure 8 shows screenshots from an example game-play of Game 6, using the parameters in Table 1. On each of the sub-figures the GUI of the human-playable game can be seen, which has the following structure:

- **Routes:** each of the three white blocks at the start of an episode represent one route (and the links it is made up from) to get from the source to the target node (as specified in the request). If a part of the block is black, it means that those spectrum slots are already occupied. In Figure 8b, it can be seen how the first two spectrum slots of the second link in the third available route are full already.
- **Selection:** the bottom row showing a green rectangle is the selection window, which represents the new requested slot. Figure 8a shows a request with a slot width of 2, whereas 8b shows a request with a slot width of 3.
- **Successful selections:** once a selection is made, it is successful if all of the slots above the selection window are free (white), and rejected if any are occupied (black).

It can be seen that the agent was performing well until the first failed allocation happened in round 10, which is when the game ended. In this case, the overall reward that agent would receive is equal to $9 \times reward_{solution} - reward_{rejection}$. However, the grid was not full by the end of the game and a different strategy could have been applied to achieve a higher reward.



(a) Game start: Round 1 **(b)** Round 2 **(c)** Round 3

(d) Round 4 **(e)** Round 5 **(f)** Round 6

(g) Round 7 **(h)** Round 8 **(i)** Round 9

(j) Round 10 **(k)** Terminal output at the start

Figure 8: Example of Game 6 gameplay

5.1.2 Game 7: Network Visualisation

Up to this point, the agent failed to outperform baseline heuristics, so we decided to implement a different structure. A top-down view of the entire network availability was given to the agent at each observation, along with a full freedom of choosing any route by creating it.

This was expected to improve the performance by creating a more unique state at any given time (since the agent has access to full network utilisation information), and hence more accurate state-action pairs. We had two ideas on how to design this:

1. Our first idea was to create a representation of the network through a table where columns represented slot numbers, and rows represented routes. For example, if a fully-connected network has three nodes (1, 2 and 3, as shown in Figure 9) and 8 slots per route, it would have 6 routes: 1-2, 1-3, 2-3, 1-2-3, 1-3-2, 2-3-1 (the capacity of each link is bidirectional, and thus 1-2 is the same as 2-1, and so on).

   Hence, a table representation of this network would have 8 columns and 6 rows. Applying the same logic, a fully-connected network with just 4 nodes (Figure 10) would result in 24 routes, and thus 24 rows in the table. It is easy to see how this representation results in an exponential increase in the size of the table as network complexity increases. Thus, this solution was discarded.

2. A more space-efficient way of representing the network is to only show the links between nodes and not entire routes, as shown in Figure 12. This way, the table representation of a network can still be reasonably sized. For a 4-node network (Figure 10), there are only 6 links. However, this poses additional issues, since multiple links need to be selected to complete a route. Therefore, it was necessary to introduce limits on what links and slots can be selected in order to adhere to the requirements.

Figure 12 shows a human-representation of Game 7 with different parameters that are stated in the sub-figure titles. It can be seen how the number of rows and columns of the table varies with with different parameters.

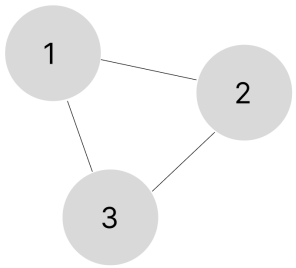


Figure 9: Game 7 network topology of a 3 node network.

The following colours were used to represent states of the slots in the network: *green* is available, *red* is unavailable, and *orange* is the current selection (which is shown for convenience and only in human-representation, as per Figure 13). Furthermore (for routes made from more than one link), after a selection of first link was made, the allowed columns would be coloured in green and rest of the columns are coloured in light green. (can be seen on Figure 15e).

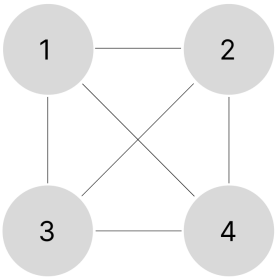


Figure 10: Game 7 network topology of a 4 node network.

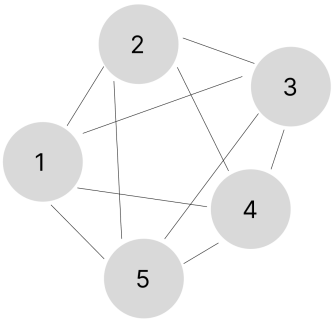
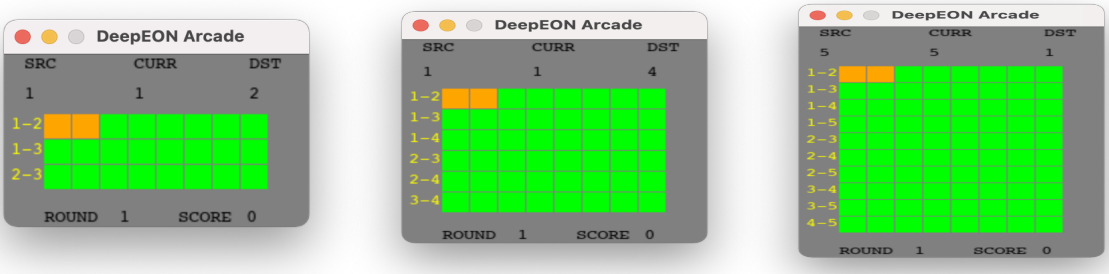


Figure 11: Game 7 network topology of a 5 node network.

Game Improvements

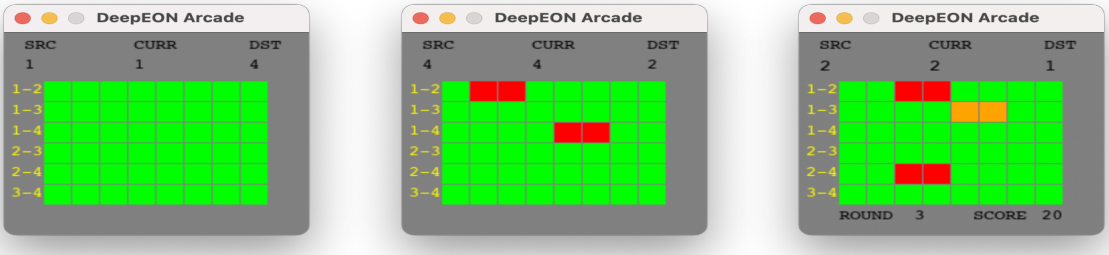
Using a new (Teleport-based) environment 3, Game 7 builds upon the previous games through the following improvements:

- **General:** there is one route allocation per round, and multiple links can be picked until the route is allocated. At the start of a round, information about a request is presented on the screen: start node, current node and end node. In human-representation, episode information is shown: round number and score. Current node has the same value as the start node at the beginning of each round and is updated as the episode progresses and links of a route are picked. Each row represents a link and has a label next to it, as shown on Figure 12.
- **Link selection:** only certain rows can be selected at any given time: *current node* has to match with either start or end of a link. For instance, in a fully-connected 4-node network, if the requested start of a route was node 2 and the first link selected was 2-4, the current node is node 4 and the next link can only be 3-4 or 1-4 (link 2-4 was already traversed). This logic has to be followed up until the last node in the selected route matches with the requested end of a route. For instance, if the *current node* was 3, requested finish node was 1 and link 1-3 was selected, the round would finish. If *current node* was 3, finish node was 1 and link 1-2 was selected, the round would not finish and *current node* would be 2. If a link is picked that is not allowed due to these constraints, the gap reward is given.
- **Slot selection:** given the continuity constraint, the slots selected for the route must be in the same position for every link in it. In this game, the action number



(a) 3 nodes. **(b)** 4 nodes. **(c)** 5 nodes.

Figure 12: Examples of Game 7 (8 slots each)



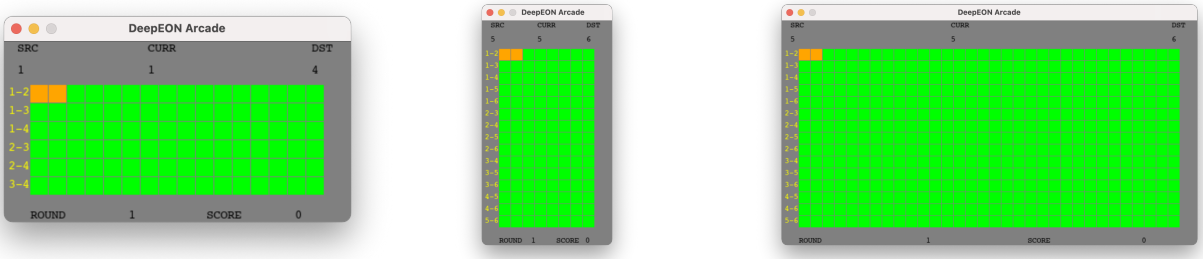
(a) Agent view. **(b)** Agent view. **(c)** Human view.

Figure 13: Game 7 version comparison

corresponds to the number of the first slot and is enforced as follows. When selecting the first slot and link of a route, any link and any slot can be picked. Once a selection is made, the number of slot selected has to be the same for the all other links of the route. For example, in a game configured to have width of 8 slots, if action 3 was selected for the first link, only actions 11, 19, 27, etc. can be selected for the subsequent links of one route. In a user-playable version of the game, entering a number was used as a selection and pressing ENTER to confirm the selection. For the agent this translated to an action space of 0 to *(rows*columns)-1* discrete actions. If the slot was already taken (represented by a red-coloured square), the allocation attempt is ignored and gap reward is given.

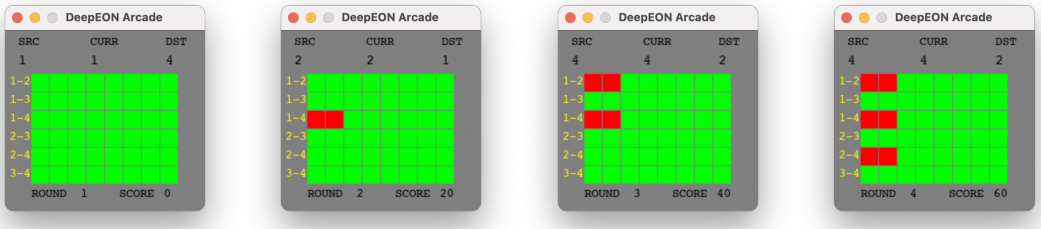
- **Finishing:** each episode is made up of multiple rounds, while each round finishes through the mechanisms described earlier. An episode finishes in one of two cases: the grid is full or a respective action is selected (with a number *rows*columns* in agent version or SPACE in a human-representation). In first case, full grid reward is given, in latter case - a rejection reward. When an episode is finished the grid gets fully reset, and no more rewards can be received for that episode.

- **Rewards:** once a round is finished, having found a route successfully, the agent receives a success reward and a new request is created, with new parameters of source and target nodes. Start and finish nodes are randomised for each request, along with

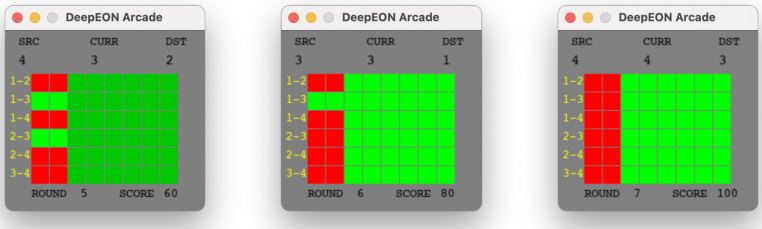


(a) 3 nodes and 16 slots. **(b)** 4 nodes and 8 slots. **(c)** 5 nodes and 32 slots.

Figure 14: Different node and slot combinations in Game 7.



(a) Round 1 **(b)** Round 2 **(c)** Round 3 **(d)** Round 4



(e) Round 5 **(f)** Round 6 **(g)** Round 7

Figure 15: Example of Game 7 gameplay.

a random slot width in ranges from 2 to 5. In order to get better agent performance, we left the slot width fixed at 2. Solution reward is given after each successful route allocation, however there was no reward for successful link allocation.

Specific rewards values used are outlined in Results section.

Figure 15 shows a Game 7 example gameplay. On each of the screenshots a GUI of the human-playable is shown after the choice was made, and hence the orange selection window is omitted. This example shows a perfect behaviour with a request allocated to using only one link on all rounds apart from one. On Round 4, source of request is node 4 and link 3-4 is picked, arriving at node 3. On Round 5, link 2-3 is picked arriving to node 2, which is the same as requested destination.

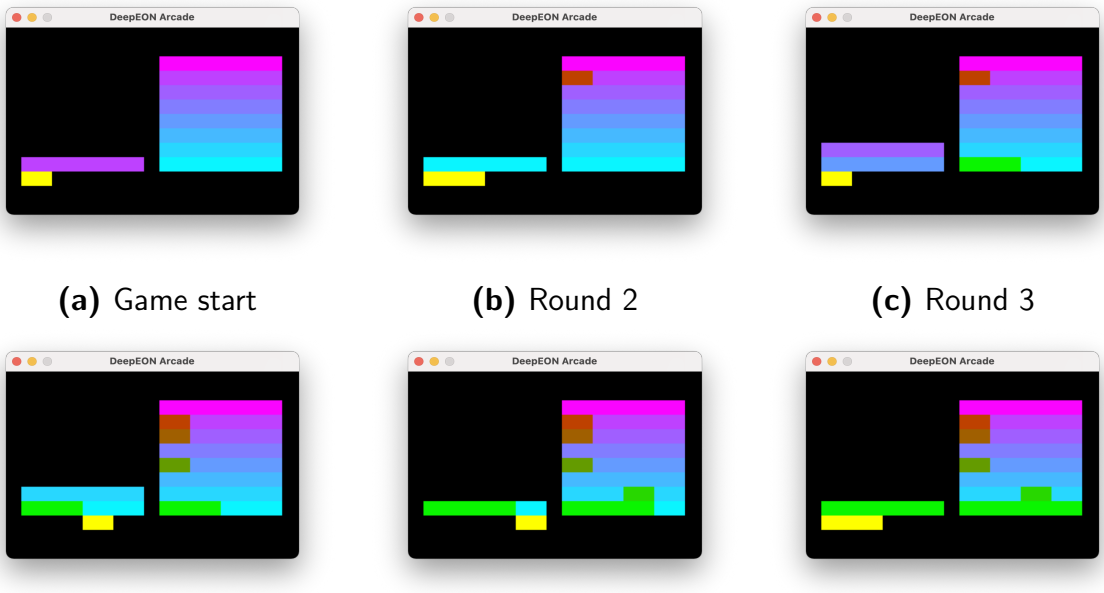
5.1.3 Game 8: Combining Improvements

Game 8 was implemented combining previous improvements to address the problem that the next state `S+1` was not consistent for each state-action pair `(s,a)`, which made the problem non-Markovian. In other words, the agent would sometimes be faced with the same GUI representing a different state, for instance showing white boxes of equal size that represent different links. This made the game mode complex and difficult to model and solve because the agent had to somehow incorporate the relevant past information into its decision-making process, making it harder for to learn a good policy that maximises the expected reward. To address this, we added a network visualisation system to show the state of network to the agent at every stage.

The game incorporates a combination of all the upgrades made in Game 6 and a version of the top-down network view implemented in Game 7. The colours are different in order to differentiate between the links, as well as tell which slots are taken. By combining this improvements, we expected an increase in agent performance with respect to Game 6 and 7.

Figure 16 shows example game-play of the human-playable version of Game 8:

- **Right side:** this side displays the status of the spectrum in the whole network, with each link being represented as a different colour. Those slots that are full are shown in a darker shade of the colour of the link.
- **Left side:** this side is home to the selection window (yellow rectangle), which varies in size (from 2 to 5) depending on the request information. On top of it, the game shows the current selected route, showing one or more links color coded as on the right side of the screen.



(a) Game start **(b)** Round 2 **(c)** Round 3

(d) Round 4 **(e)** Round 5 **(f)** Round 6 (Block)

Figure 16: Example of game 8 gameplay (16 slots, end on first failure)

5.1.4 Improving DQN

Despite the implementation of new games, the agent failed to outperform the heuristics on any of the games (as shown in Section 6.1). Since the performance was not satisfactory, an improvement was sought in the DeepRL algorithm itself.

Apart from performing hyperparameter optimisation, we also looked at algorithms that boost DQN performance (Double Q Learning, Dueling DQN, etc). However, in order to gain multiple benefits all at once we decided to use an algorithm that was built on top of multiple enhancements to DQN.

Rainbow [25] is a DeepRL algorithm that brings together multiple improvements in DQN. It was released in 2017 and showed state-of-the-art performance on the Atari 2600 benchmark. Rainbow combines the following DQN improvements: Double DQN for addressing overestimation of action values, Prioritized Experience Replay for more effective learning from important experiences, Dueling Network Architecture for better representation of state values and action advantages, multi-step Learning for capturing the effect of action sequences, Distributional RL for modeling the distribution of returns, and Noisy Nets for injecting parameter noise to encourage exploration, resulting in a powerful and efficient RL algorithm.

By using these improvements, Rainbow achieved state-of-the-art performance on a variety of visual games [25]. We hoped that using Rainbow for our project would improve the performance of the agent. However, it proved to be too difficult to integrate Rainbow into the existing project and the idea was abandoned.

5.1.5 Dreamer V3 World Model

During the research on improving the performance of DQN, we decided to try a model-based approach and see if the performance increased. The key difference between DQN (or Rainbow) DeepRL algorithm and a world model is that a world model relies on learning an internal model of the environment to plan actions, while DQN directly learns the action-value function without the need for an internal model. We decided to use the newly-released world model Dreamer V3 [26], which was published in early 2023, and therefore unavailable at the start of this research in 2022.

DreamerV3 is the latest generation of Dreamer algorithms, focusing on visual control tasks in complex and dynamic environments. The original Dreamer was published in 2019 [27], with versions 2 and 3 showing significant improvements in performance [26]. The algorithm works by learning a world model from experiences, using a combination of latent imagination, model-based planning and learning from imagined trajectories. Dreamer V3 uses actor-critic architecture where the actor (policy network) learns to take actions and the critic (value network) learns to evaluate those actions.

DreamerV3 demonstrated state-of-the-art performance on a wide range of challenging visual control tasks with total independence, outperforming previous model-based and model-free algorithms. For instance, DreamerV3 is the first algorithm to collect diamonds in Minecraft, starting from scratch without utilising human data or curricula, a task that has been a long-standing challenge in the field of artificial intelligence [26].

It was expected that DreamerV3 would be able to solve the RSA game and outperform the heuristics since this was another visual challenge. Game 8 was used, since it incorporated all of the improvements developed throughout this paper.

5.2 Routing Game

Prior research has shown that splitting the RSA problem into routing and spectrum allocation sub-problems can enable a reinforcement learning agent to outperform heuristic algorithms in resource allocation for Elastic Optical Networks (EON) [9], [11], [12]. In this context, the application of Deep Reinforcement Learning (DeepRL) for solving the Routing problem in EON has been explored, utilising a visual representation for the environment. This section outlines the pipeline for developing the routing game, which encompasses a sequence of key steps, including environment design, model implementation, training, evaluation, and policy analysis.

5.2.1 Environment Design

Various design methodologies and requirements were considered for designing the environment for the DeepRL agent. It was important that the game would provide a visual representation of the EON, while also modelling the requests that need to be allocated and some representation of the availability of network resources. In order to achieve a dynamic and engaging learning experience, the game would need to be interactive and provide feedback on decisions made by the agent. To provide real-time availability for the network resources, the game should implement the first fit heuristic for spectrum allocation which was described in Section 4.1. Furthermore, the action space for the agent was limited to a small and fixed set of choices with a strong focus on consistency, to promote playability and effective learning.

The RSA environment was simulated using two types of games: one for human players which was used to evaluate functioning of all systems and exploration of the network, and one of the DeepRL agent which was modelled on the former. In total, two human games and two equivalent agent games were implemented, each incorporating different features or strategies for solving the same problem. In the following sections, we will describe different versions that were developed, with a focus on the agent game.

As in the RSA game [22], the developed games are based on a technology stack that utilises various Python libraries, including *arcade*, *networkX*, *pygame*, amongst others. The *arcade* library has been used for implementing the game, while *networkX* is used for creating the topology of the network. Additionally, other Python libraries have been employed to add functionality and enhance the performance of the code.

In the following sections we will describe different versions of the games that were developed with a focus on the agent game.

Routing Game

The human game was designed to give the user an understanding of the underlying mechanics of the system, including network traversal and the impact of the route on the global spectrum. Figure 17a shows the GUI of the human game.

Each round consists of a request, made up of a source node, a target node, and a number of slots to be allocated, as shown in Figure 17a. An episode continues going through rounds until the agent selects an unavailable node 4 times (blocks), when the score for an episode is saved and compared against the global high score. A node is available if it meets three conditions:

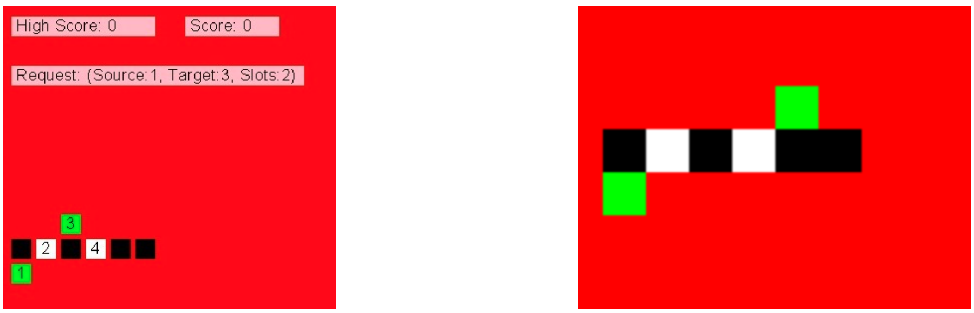
1. A link is available between the node and the current node.

2. There are empty contiguous slots available equal to the request size in the link between the two nodes.

3. The node has not been visited before along the route.

The three boxes at the top of the human game display information about these metrics. At the bottom, there are three rows of boxes:

- **Top row:** shows a green box in the position, and number, of the target node (in this case 3).
- **Middle row:** shows 6 boxes, one for each node in the network. The white boxes represent the nodes that are available to the user to move next to from the current node (nodes 2 and 4). Black boxes are unavailable nodes, and thus inaccessible from the current position (nodes 1, 3, 5 and 6).
- **Bottom row:** shows a green box in the position, and number, of the node the user is currently at (in this case 1).



(a) Human game **(b)** Agent game

Figure 17: Routing game

The game works by making the user build a route (made up of a list of links between nodes) from the source node to the target node by moving from the current node (in the bottom row) to one of the accessible nodes (in the middle row). This is repeated until they the target node (shown in the top row) is accessible from the current. The agent game (Figure 17b) works just like the human game, but without the three metrics boxes and the numbers in the node boxes. This is to simplify the visual representation and avoid confusing the agent.

During the development of the game, we started with an arrows-based movement (like in Section 5.1.1), whereby action space consisted of three options: moving left, moving right, and pressing enter to move to the node selected. In this manner, the user would press LEFT and RIGHT to move the current node (bottom row) to under the desired next node (middle row) and clicking ENTER to move forward.

However, from the RSA game (as developed in Section 5.1.1) we noticed this could be a non-optimal action space, given that mapping rewards to actions that were effectively non-consequential (just moving left and right) could confuse the agent. Thus, we decided to implement a teleport option, changing the action space to selecting the number of the

node to jump onto. This way, if the user wants to go from node 2 to node 5, it would only require one action (using up one timestep), instead of 4 (moving right 3 times and pressing enter).

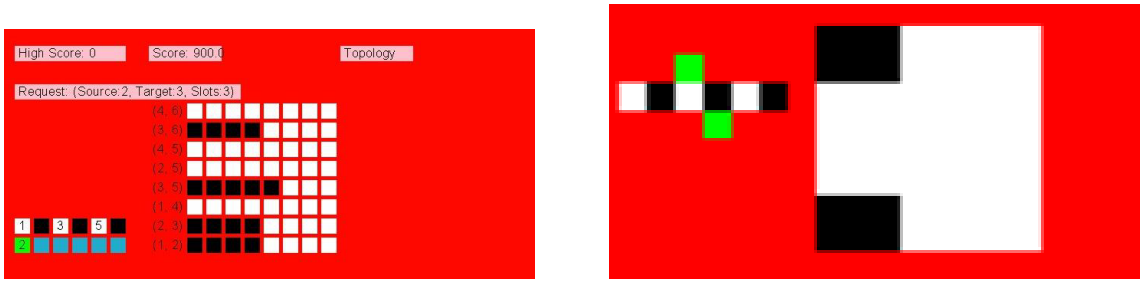
Once the agent game was fully developed, we exported the mechanics to an agent-friendly GUI (Figure 17b), simplifying the interface by removing the request bar, numbers in the boxes, and putting all boxes together.

Routing Game with Network Visualisation

In the routing game presented above, the agent can only consider the availability of nodes in the network from its current position when choosing the next node to move to, without any knowledge of the available spectrum for links or an understanding of the resources allocated throughout the network. The allocation of spectrum in the network, carried out by the heuristic, was unavailable to the agent, but it still impacted the environment state. As a result, an abstraction layer that can limit the agent's decision-making process is introduced. This is a similar problem and solution as developed in Section 5.1.3.

To address this problem, a solution can be to provide the agent with an additional view of the spectrum allocation in the entire network. Therefore, the performance could be improved because information about the general context of the problem is available.

The network visualisation is implemented by adding a grid on the right side of the game. Each row of the grid represents the current state of the spectrum slots for one link in the network, as shown in Figure 18. A square on the grid is white if the spectrum slot is available and black if it is allocated. In this context, only blocked resources for requests that have already been allocated are shown as black since a route for a request is selected over multiple actions. Therefore, the grid is updated only after all the slots in a route are selected and the route is validated. The network state implantation for the human game is shown in Figure 18a and for the agent game in Figure 18b.

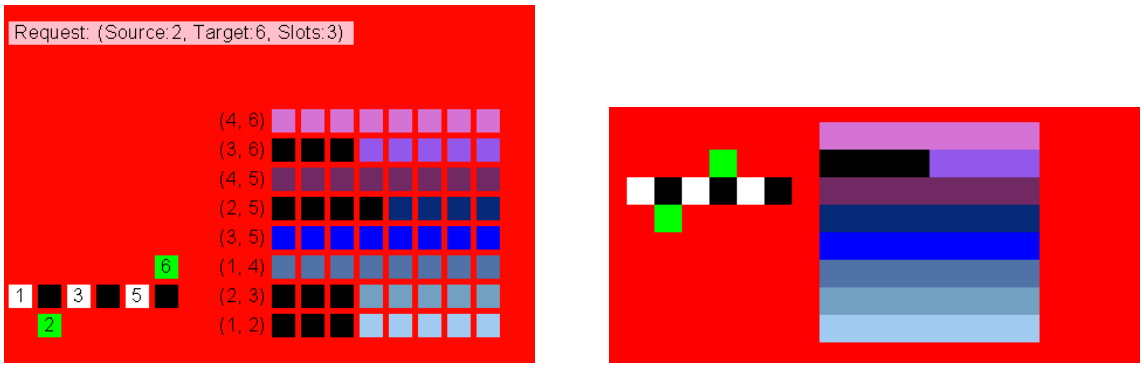


(a) Human game **(b)** Agent game

Figure 18: Network visualisation

An alternative proposal for network representation involved using different colors to represent free slots from each link in the network, as developed for Game 8 in 5.1.3. Therefore, each link is distinguished by a color (Figure 19). However, this change adds complexity without necessarily providing more information to the agent, so it was not included in the final version of the agent game.

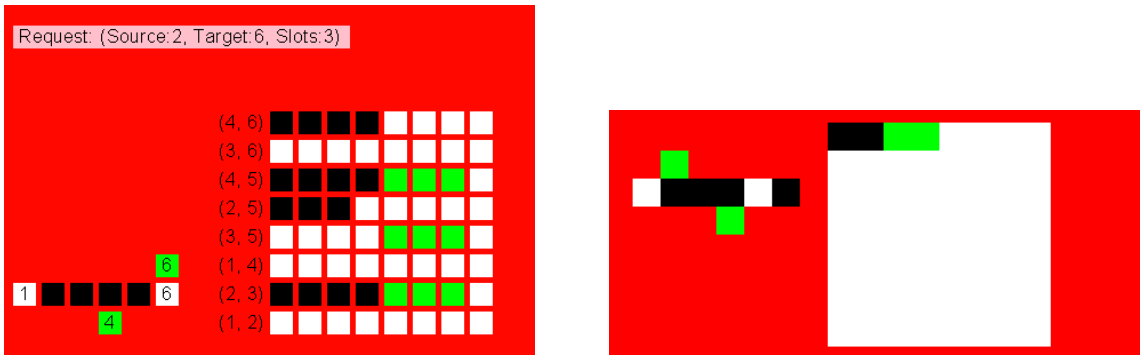
To improve the representation further, green squares were used to indicate the hypothetical impact on the network spectrum based on the agents current node selection.



(a) Human game **(b)** Agent game

Figure 19: Network visualisation with colors

Previously, the network representation would change only after a request had been completed and the agent received a solution reward. In the updated representation, the agent is provided with more information on the consequences of each taken action. This is the final version for visual representation and it is shown in Figure 20.



(a) Human game **(b)** Agent game

Figure 20: Network visualisation with spectrum representation along the route

The visual representation of the network was implemented in both the human game for testing purposes and the agent game to provide additional information to the agent. While testing the network visual representation on the human game, we identified an issue with the spectrum allocation algorithm. The issue occurred when the `check_with_history()` function was used. When choosing a new node, the availability of the node is assessed based on whether there is available spectrum on the link between the current and the new node. This process involves comparing the spectrum availability on the link grid with the spectrum allocation history of previously selected links on the route. The `check_with_history()` function, which is responsible for checking spectrum availability on the links already selected in the route, was not working, causing requests to be allocated on spectrum that is already used. As a result, some of the spectrum slots were being allocated twice, which gave the agent an unfair advantage compared to the heuristic. We solved the problem by rewriting the function and recomputing all the results to ensure fairness.

Technical Implementation of Games

This section details the code structure and the special data structures used for tracking the network state and spectrum allocation. The game objective is to allow the player to solve only the routing problem while the spectrum is allocated using First Fit algorithm. In order to simplify the game, the player selects only one node in the route at a time. However, this introduces a level of complexity in the integration of spectrum allocation within the game since the spectrum slots can be blocked within the network only after the whole route was selected.

The game keeps track of the current node and receives from the player a next node to move at. The availability of all nodes in the network from the current node is presented to the player, before they make the selection (as described in this Section).

To keep track of the spectrum availability both during the request allocation and across multiple requests, and to ensure the spectrum is allocated following the First Fit algorithm, two custom data structures are use: link grid and Running Path Spectrum (RPS). More information on how these two work fundamentally can be found in Appendix A.

Moreover, a more in depth view of the overall implementation of the agent game an its integration with custom environments can be found in Appendix B.

5.2.2 Model Implementation

This section details the methodology used to implement a DeepRL agent for playing the game developed in section 5.2.1. The reinforcement learning agent used is Deep Q Network (DQN) from Stable Baselines 3. A custom environment was created using the Open AI Gym library to provide interaction between the agent and the environment.

The custom environment is a Python class that provides a frame for defining the observation and action space, rewards received by the agent, and environment changes rules. Three custom environments were developed for the agent to play one of the two games: routing game or routing game with network visualisation. Figure 21 shows the three versions of the environments on which the agent is trained:

- **Version 1:** Custom environment 2 implements the arrows-based movement for selecting a node and uses the routing game. The action space is defined by the following 3 actions: LEFT, RIGHT and ENTER.
- **Version 2:** Custom environment 3 implements the teleport option for selecting a node and uses the routing game. The action space is represented by the nodes available in the network. For the network topology provided the action space is 6 consisting of 0, 1, 2, 3, 4 and 5.
- **Version 3:** Custom environment 4 implements same teleport option, but uses the routing game with network visualisation. The same action space as in Version 2 is employed.

After the agent takes an action, a snapshot of the game is returned as an observation, resulting in an RGB observation space with the size of the game window ($screenwidth \times screenheight$). The observation space remains the same across all versions.

All custom environments have a similar structure, with the following methods:

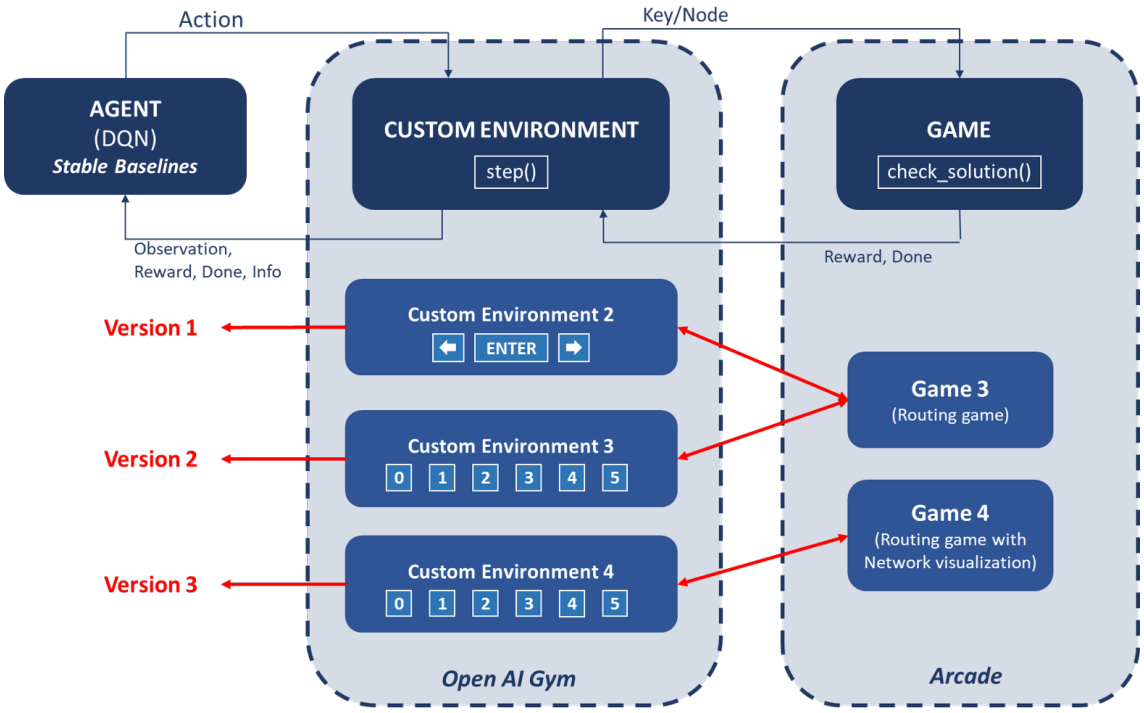


Figure 21: Model implementation

- `init()`: This method is used to define the action and observation space and to instantiate the game that will be played by the agent.

- `step()`: This method is used by the agent to interact with the environment as it can be seen in Figure 21. An integer,represents the action selected by the agent, between 0 and 2 for arrows-based movement or between 0 and 5 for teleport option is passed to the function by the agent. The integer is then converted to the corresponding node, and the `check_solution()` method from the game is called. This applies the corresponding changes to the environment, and returns the reward and a Boolean variable 'done' that is set to True when an episode is finished. Then, the observation is updated by rendering the window of the game. Finally, the method returns the observation, reward, Boolean variable and additional information.

- `reset()`: This method is used at the end of each episode and is called when the Boolean variable 'done' is set to True. It draws a new round of the game and returns an observation with the state of the environment at the beginning of the episode.

- `render()`: This method is used to render the environment for recording of the screen or open a game window for visual inspection.

- `close()`: This method is used to exit the game.

5.2.3 Training and Evaluation

To train the DQN model we used the Stable Baselines 3 library learn function, adding a Weights and Biases (WandB) callback to enhance our analysis. WandB is a machine learning platform useful for collaborating, tracking experiments, visualising and evaluating training performance, and performing hyperparameter optimisation.

The first step, however, was to set up the training platform at UCL Electronic & Electrical Engineering department's servers, to make use of the increased computational capacity and drastically decrease training time, when compared to our personal computers. Once we managed to get the secure shell (ssh) connection up and running, we began experimenting with training parameters.

The main metric chosen to evaluate the performance of the agent was the average reward per episode (game). We chose this metric because we wanted to push the agent to try to outperform heuristics first and foremost, even at the expense of reliability or variance.

Once our metric was selected, and the training infrastructure set up, we used WandB to do hyperparameter optimisation for both the normal and network visualisation games. We did this through WandB's sweep tool, which allowed us to search through a predetermined set of hyperparameters using a Bayesian optimisation algorithm. This algorithm builds a probabilistic model of the objective function, using it to guide its search balancing exploration and exploitation, resulting in faster convergence. Table 2 shows the effect of each hyperparameter tested.

Parameter	**Function**
`batch_size`	Number of experiences sampled from the replay buffer to update the DeepRL model in each training iteration.
`buffer_size`	Maximum replay buffer capacity, which stores past experiences for training the DeepRL model.
`exploration_final_eps`	Final exploration rate value (epsilon) used in an epsilon-greedy policy during training, which determines the probability of selecting a random action.
`exploration_fraction`	Fraction of the total training time over which the exploration rate (epsilon) is normalised from its initial value to its final value.
`gamma`	Discount factor that determines the weight of future rewards. Zero represents immediate rewards and one represents equal weight given to immediate and future rewards.
`learning_rate`	Rate at which the model's parameters are updated based on the calculated gradients during training.
`learning_starts`	Number of timesteps before starting to update the model through sampling experiences from the replay buffer.
`target_update_interval`	Frequency at which the target network is updated with the weights of the main network during training.
`train_freq`	Number of timesteps between successive updates of the DeepRL model by sampling experiences from the replay buffer.

Table 2: Hyperparameter overview

After obtaining the best-performing hyperparameters, we implemented them in the

game's config file and trained the agent as explained earlier.

5.2.4 Policy Analysis

The last objective of the project is to understand the agent's behaviors by using elementary explainability methods. This section provides an overview of the methods used to extract information from the evaluation of the agent. The obtained data is used to further understand and interpret the performance of the DQN agent in solving the RSA problem.

An optimized policy is used by the agent during evaluation. Therefore, the data used for exaplainabilty was collected from the evaluation of the agent. To understand the policy employed by the agent, the information about each action taken by the agent during evaluation is extracted. To make this information available, the following metrics have been implemented across all custom environments: action ID, action performed, and reward received. To be able to interpret the actions of the agent, data that provides information about the context in which these actions have been taken is required. The agent is given an environment which captures information about the task that has to be solved and the state of the network. Given that the agent's task involves solving new requests, this process involved modifying both versions of the game to capture and return information regarding each request, including the request ID, source, target, and number of slots.

All the information above are useful in the context of episodes. Hence, a custom evaluation was developed to systematically gather and store this information in a series of data frames, thereby enabling further analysis and computation.

Three additional analytics have been calculated based on the gathered data:

- *Episode length:* number of actions taken within an episode (episode-level metric).
- *Route length:* number of links in each completed route, once a request has been fulfilled or the episode has ended (request-level metric).
- *Network utility:* percentage of the network that has been filled in the entire network by the end of the episode. There are 8 available slots in 8 different links resulting in a total network capacity of 64 slots (episode-level metric).
- *Episode rewards:* rewards at the end of each episode (episode-level metric).

6 Results and Discussion

6.1 RSA Game

6.1.1 Game Optimisations

In this project, one of the approaches to improving the performance of the agent was implementing various improvements to the game. In this section, the results of these experiments are shown and analysed.

Environment 1 versus 2

The first improvement experimented with was changing the action space from left and right arrows (env 1) to an action space where each action represents a position in the spectrum (env 2). The two environments were compared against each other with all other

variables remaining constant. See Figure 22 which depicts the agent's score over 1000 episodes.

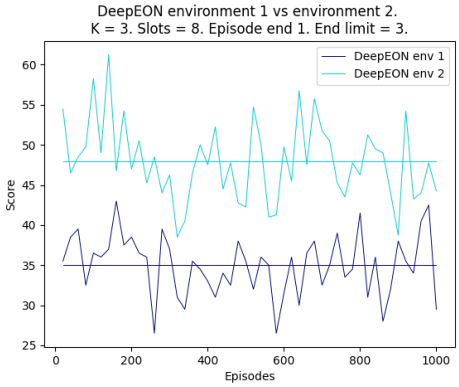
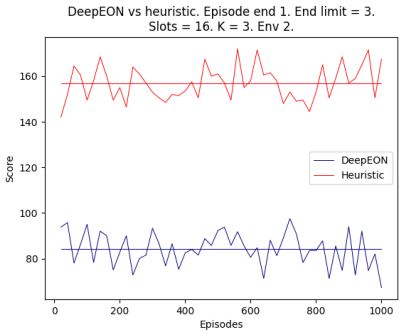


Figure 22: Environment 1 (arrows) vs Environment 2 (teleportation).

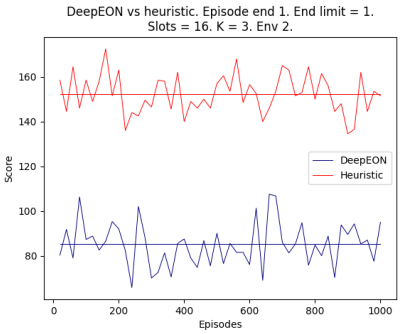
It is apparent from the figure above that Env 2 demonstrates superior performance than Env 1 by 37.1%. This could be explained by the reduction of the effects of the sparse reward problem, which is the challenge of training a reinforcement learning agent when it receives only occasional and delayed feedback signals for its actions. Mapping each action to a spectrum slot reduces the delay between action and reward.

Episode Ending

The next experiment conducted was to trial a different episode-ending mechanism. The current mechanism ends the episode after n rejected allocations. The new suggested mechanism ends the episode after a fixed number of timesteps, regardless of the number of rejected allocations. The hypothesis was that this would improve the performance since it reduces the effect of the agent making a mistake, something that the heuristic is immune to. See Fig 24. We also experimented with setting n in the current mechanism to different values. See Figure 23a and 23b.



(a) Agent (STD: 38.46) vs heuristic (STD: 33.35). Heuristic performed 86.3% better.



(b) Agent (STD: 47.97) vs heuristic (STD: 33.96). Heuristic performed 78.9% better.

Figure 23: Ending mechanism 1. End limit 3 (left). End limit 1 (right)

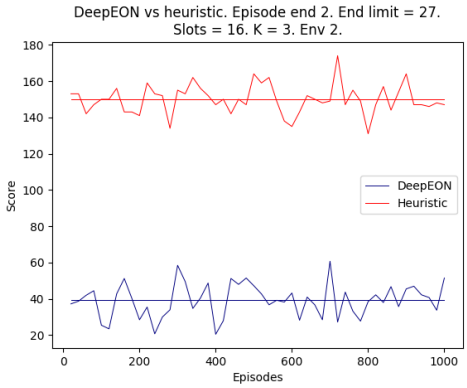
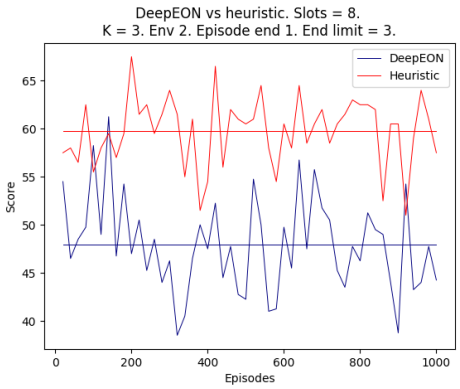


Figure 24: Ending mechanism 2. End limit 27. Agent (STD: 39.52) vs heuristic (STD: 40.78). Heuristic performed 282.0% better.

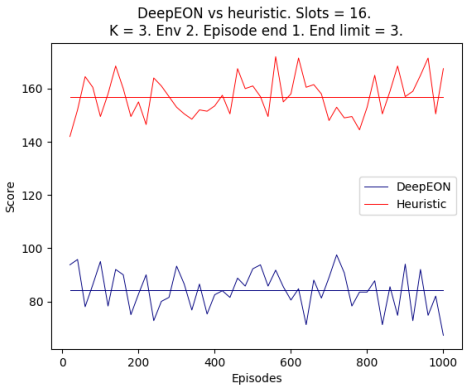
The results from these experiments show that the current ending mechanism yields the best performance. The heuristic performed 282% better than the agent with the next ending mechanism compared with 78% with the current. The results of changing the value of n show that the performance of the agent drops by 7.4% when n is changed from 1 to 3. This result does not necessarily indicate that the agent's learning was affected, it could be due to the fact that when n=3 the episode lasts longer, allowing more opportunity for the heuristic to outperform the agent, resulting in a higher percentage advantage.

Number of Slots

Next, the number of slots available in the spectrum were experimented with to see if this might improve the performance of the agent. The assumption was that with a larger spectrum, the agent has more options to develop novel techniques to outperform the heuristic, which is restricted to a first-fit policy. See Figure 25a, 25b and 26.



(a) 8 slots. Agent (STD: 23.41) vs heuristic (STD: 20.11). Heuristic is 36.3% better.



(b) 16 slots. Agent (STD: 38.46) vs heuristic (STD: 33.34). Heuristic is 86.3% better.

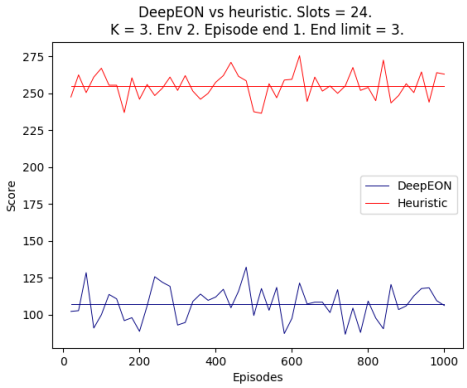


Figure 26: 24 slots. Agent (STD: 52.16) vs heuristic (STD: 41.86). The Heuristic is 137.5% better.

The results show that as the number of slots increases, the heuristic outperforms the agent by a higher percentage. See Table 3 It can be deduced that there is a linear negative relationship described by the equation:

$$y = 6.25x - 13.7,$$
where x = number of slots
y = percentage of performance difference for heuristics

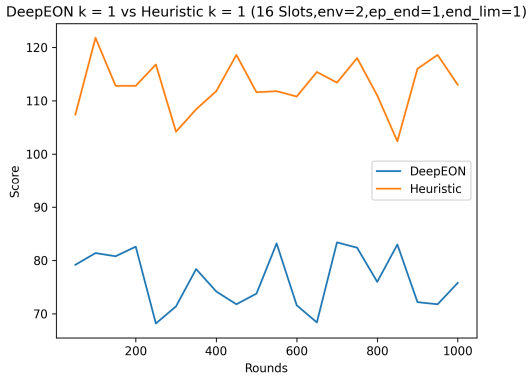
These might also be due to the fact that increasing number of slots will also increase the episode length and therefore heuristic has more opportunity to outperform the agent.

Slots	% outperformance
8	36.3 %
16	86.3 %
24	137.5 %

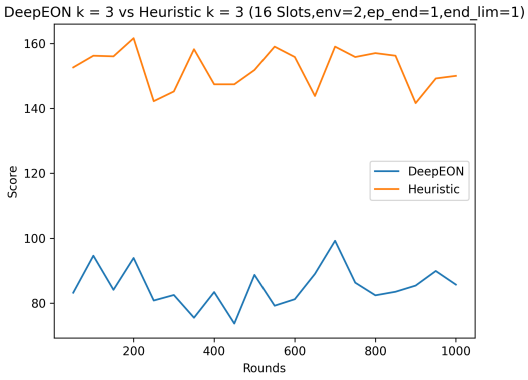
Table 3: Slots vs percentage that the heuristic outperformed the agent

Removing Routing

Next, the routing part of the problem was removed by setting K=1, with the expectation that this would simplify the problem and result in an increased agent performance. Figs 27a and 27b show the results of heuristic Vs agent in a k=1 scenario and also k=3 scenario.



(a) Heuristic performed 47.5% better.



(b) Heuristic performed 78.9% better.

Figure 27: Agent vs Heuristic. K = 1 on the left, K = 3 on the right

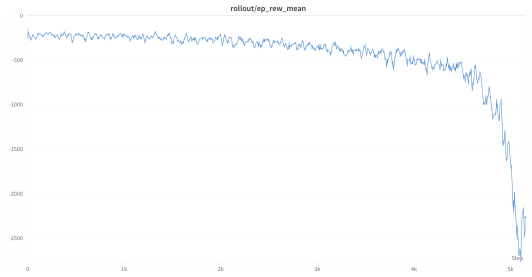
As we can see, the heuristic only outperformed the agents by 47.5% when k=1 as opposed to 78% with k=3. This could be due to a simplification of the problem but is also likely due to the fact that when k=1, the average episode length is reduced by around 15%.

6.1.2 Game 7

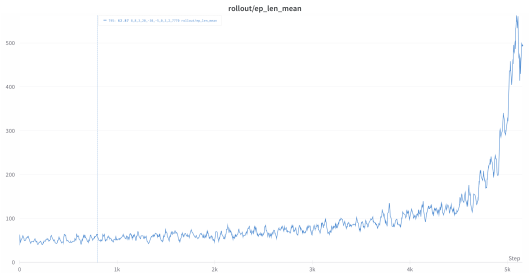
Various combinations of parameters were tested for Game 7, and a combination yielding the best results can be seen on Table 4. The parameters were selected by hand and it is likely that running a sweep would significantly improve performance.

In order to better see the progress of the agent, a GIF 10 episodes was added to WandB platform every 100,000 episodes (out of the total 2,000,000), which can be seen on Figure 29.

Due to not having an available heuristic algorithm, the performance could only be measured by graphs provided by WandB such as mean episode reward, shown on Figure 28a and mean episode length shown on Figure 28b. Despite the initial improvements, mean episode reward decreased and mean episode length increased as training progressed. This was consistent across all training attempts and could be due to the game being too complex.



(a) Mean episode reward.



(b) Mean episode length.

Figure 28: Game 7 training (2M steps).

Parameter	Value
environment	3
number of nodes	4
number of slots	8
request slot width	2
solution reward	20
full grid reward	1000
rejection reward	-50
gap reward	-1
total timesteps	2,000,000
learning rate	0.0001
batch size	64
buffer size	100,000
target update interval	20,000
training frequency	4 steps

Table 4: Game 7 example parameters.

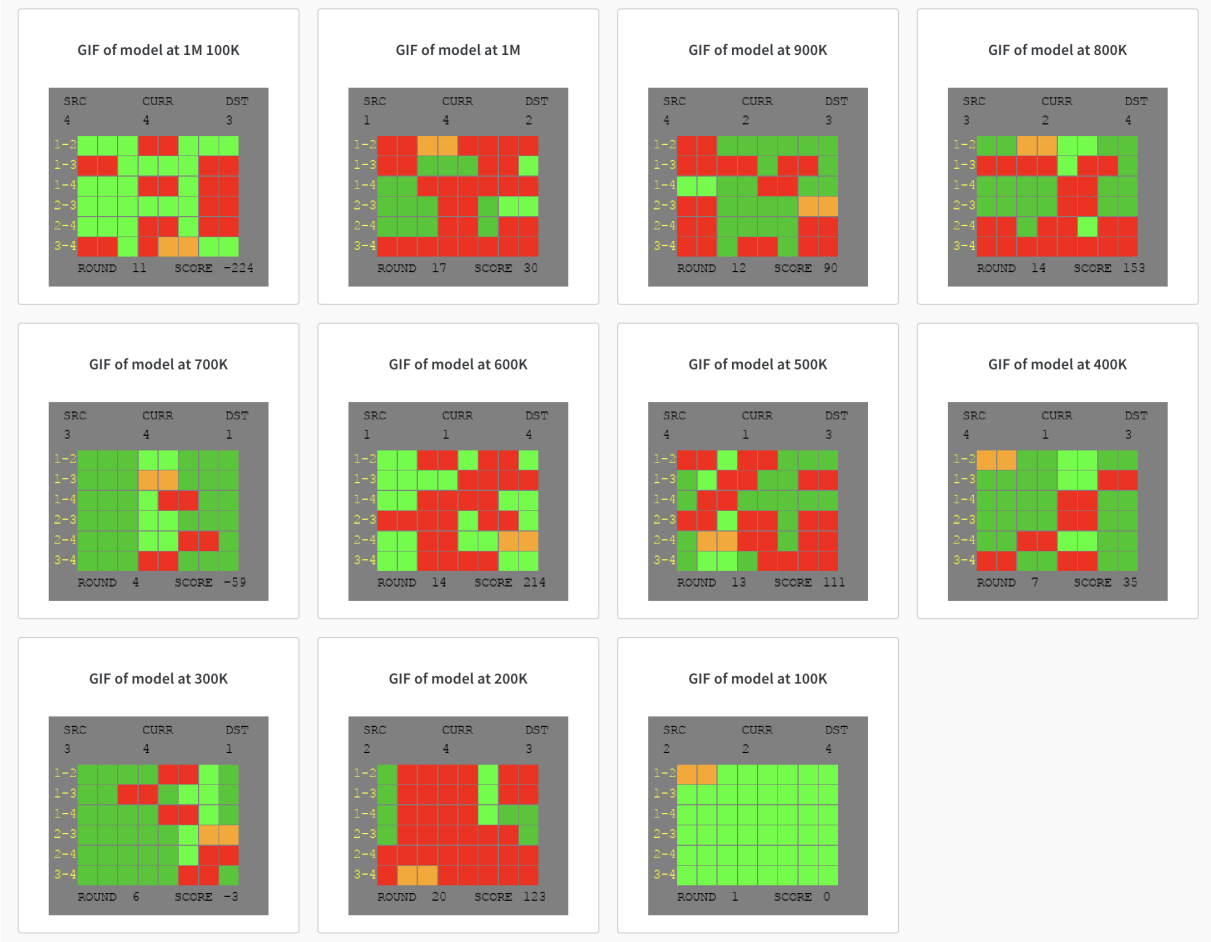
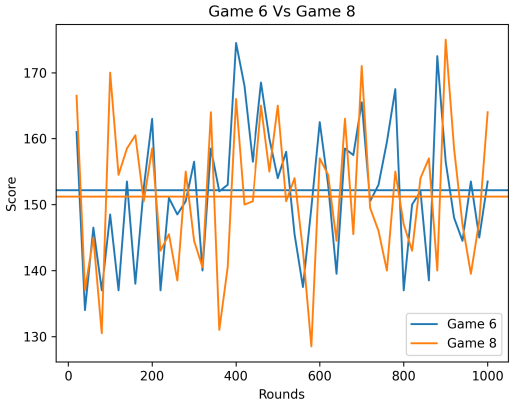
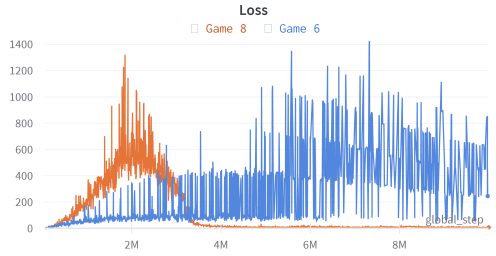


Figure 29: Tracking Game 7 training on WandB.

6.1.3 Game 8



(a) Game 6 vs Game 8 performance



(b) Training loss Game 6 vs Game 8

We can see from Figure 30b that Game 8 resulted in a convergence of the training loss compared with Game 6, which doesn't converge. This is because game 6 is non-markovian, so it would be impossible to converge. Unfortunately, game 8 did not improve the performance of the agent, and in fact, we saw a drop in performance, see Fig 30a. This is quite a surprising result since it would make sense for the agent whose loss function converged to perform better.

6.1.4 Dreamer

The final experiment conducted was to test the DreamerV3 model on game 8 to see if it could beat the performance demonstrated by the current DQN agent and also the heuristic. See Figure 31 which compares the results of the heuristic against the DQN agent in games 6 and 8 and Dreamer in game 8.

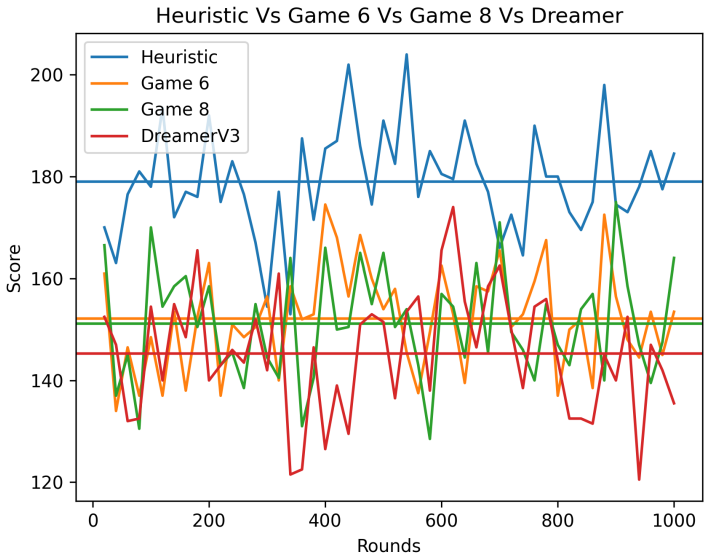


Figure 31: Game 8 vs DreamerV3

We can see that Dreamer failed to improve upon the DQN's performance against the heuristic. Future research could look at fine-tuning Dreamer for the purposes of this

project to see if the performance can be improved. The results from Dreamer have thus far be captured from Dreamer out of the box with no alterations of optimisations. Di

6.2 Routing Game

6.2.1 Initial Results

Both agents (using teleport and arrow action spaces) were trained on the 6-slot routing game version 3 using 500k timesteps. The initial results of the project can be seen in Table 5 and Figure 32. The figure illustrates that both cases resulted in improved average performance over 10000 episodes, compared to the First Fit heuristics, albeit with higher variance. In particular, the teleport action space is significantly better than arrows, similar to the RSA game.

	Average Reward		Standard Deviation	
Model	Absolute	vs Heuristic	Absolute	vs Heuristic
Heuristics	33.10	NA	19.08	NA
Game 3	39.95	+20.69%	24.27	+27.2%
Game 4	42.57	+58.82%	22.43	+17.56%

Table 5: Initial agent performance versus heuristics

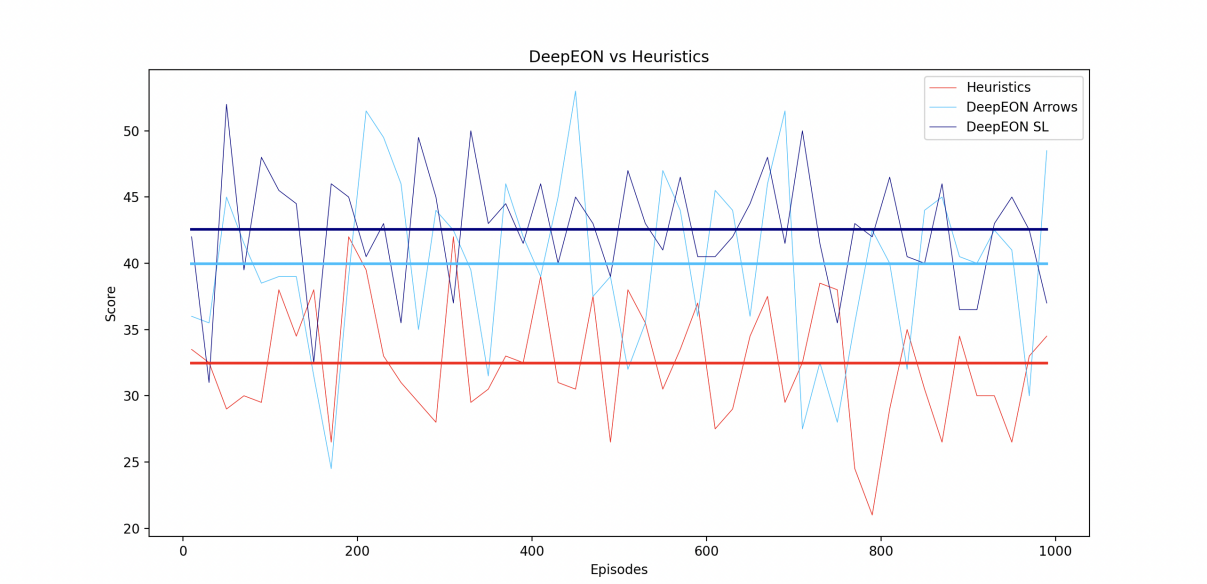
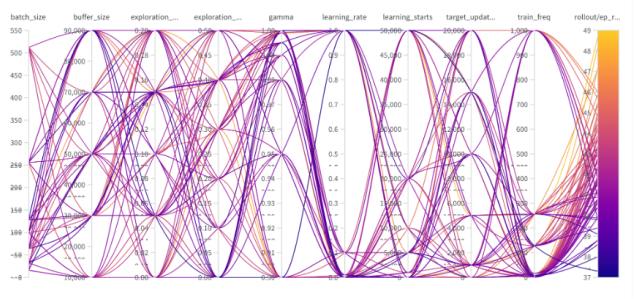


Figure 32: Initial results: 20-episode average score for agent and heuristics

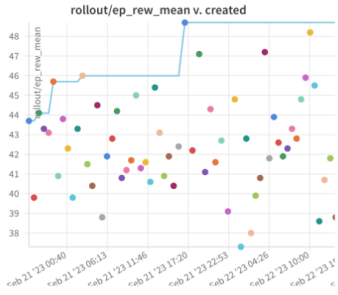
To improve performance and reduce variance, a hyperparameter sweep was conducted using WandB, as depicted in Figure 33. The optimized DQL agent achieved an average mean of 48.7, resulting in a performance improvement of 47.13% compared to the heuristic, and 14.4% higher compared to the unoptimized agent.

With these promising results, our team shifted focus towards enhancing the explainability of the model and attempting to reverse-engineer the routing policy of the optimized DQL agent.

Given the better performance of teleport compared to arrows, we only kept analysing its performance as we made more changes. The development of game 4 involved incorporating



(a) Hyperparameter breakdown



(b) Average episode reward

Figure 33: Initial sweep

a visualization approach to display the network state on-screen, based on the teleport action space, with the aim of further improving performance and facilitating explainability efforts.

Upon further investigation into the model's behaviour, we discovered a bug in the game process that allowed for the allocation of occupied slots in the network under certain circumstances.

6.2.2 Bug Fix

Upon meticulous examination of the code, it was identified that the bug stemmed from an incorrect implementation of the `check_with_history()` function resulting in false returns for the `check_spectrum()` function. This bug bestowed the DQL agent with an unfair advantage, making previous results invalid thus, the training and evaluation needed to be repeated. Although this incident was unfortunate, it serves as a compelling example of the importance of post-analysis of results and the benefits of having a visual representation of the model's behaviour.

	Average Reward		Standard Deviation	
Model	Absolute	vs Heuristic	Absolute	vs Heuristic
Heuristics	33.10	NA	19.08	NA
Game 3	30.58	-7.57%	22.28	+18.64%
Game 4	10.56	-68.09%	16.73	-10.92%

Table 6: Agent performance vs heuristic after fixing the bug

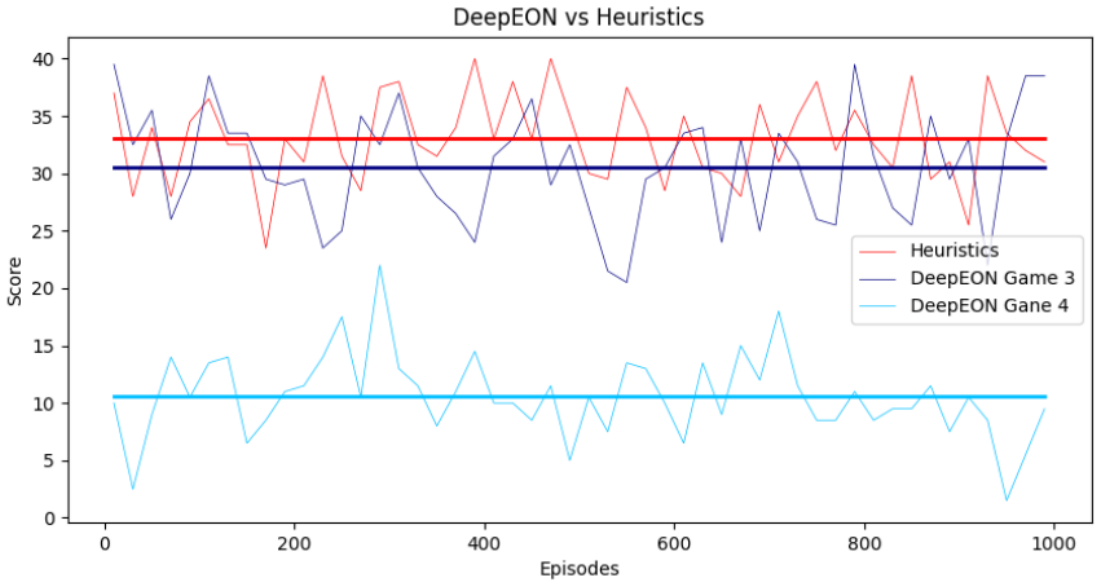
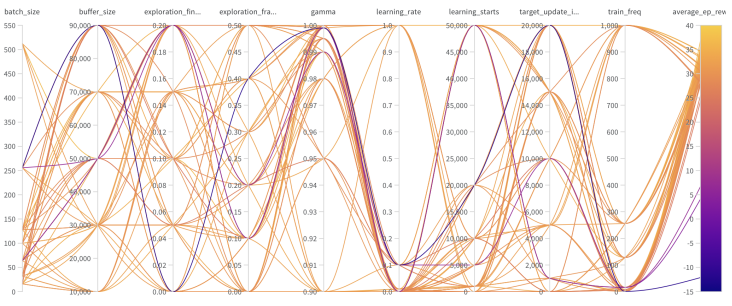


Figure 34: Results after bug fix: 20-episode average score for agent and heuristics

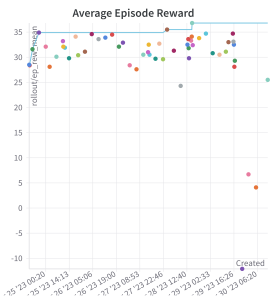
The performance of the models after addressing the bug using 500k timesteps for retraining is illustrated in Table 6 and Figure 34. game 4's performance was unexpectedly worse than game 3, despite providing the agent with additional information. The increased complexity of information in game 4 is a possible reason for this outcome. It is evident that the agents no longer outperform the heuristic. Nevertheless, game 3 demonstrates a close performance, with only a 7.57% difference. Considering the significant 14.4% performance boost achieved earlier through hyperparameter tuning, these results were still encouraging.

6.2.3 Final Results

We performed a WandB sweep for both the routing game (Figure 35) and the game with network visualisation (Figure 36) using 200k timesteps to find the optimal parameters for each. In both cases, the sweeps seem to end up converging near the optimal performance parameters, even though the network visualisation game had a small cluster of runs ending up with highly negative values, which does not seem an issue given the performance of the rest of them. The final hyperparameters chosen are shown in Table 7.

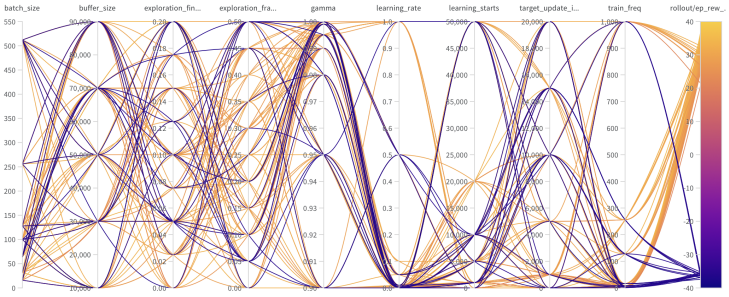


(a) Hyperparameter breakdown

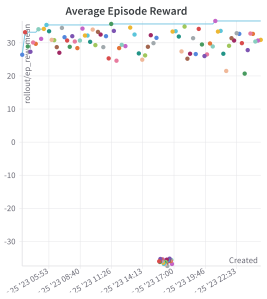


(b) Performance overview

Figure 35: Game 3 sweep



(a) Hyperparameter breakdown



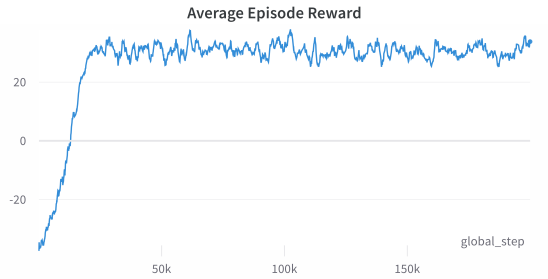
(b) Performance overview

Figure 36: Game 4 sweep

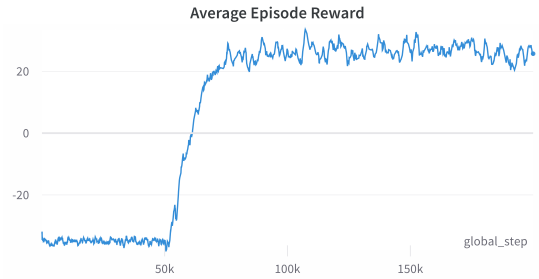
Parameter	Game 3	Game 4
`batch_size`	512	100
`buffer_size`	50,000	70,000
`exploration_final_eps`	0.1	0.2
`exploration_fraction`	0.1	0.15
`gamma`	0.99	0.99
`learning_rate`	0.01	0.00001
`learning_starts`	1,000	50,000
`target_update_interval`	1,000	20,000
`train_freq`	4	16

Table 7: Optimised hyperparameters

With the optimal hyperparameters selected, we trained and saved the new models using 200k timesteps, which converged quickly as shown in Figure 37.



(a) Game 3



(b) Game 4

Figure 37: Training episode reward evolution

We then loaded the model and evaluated it for 10k episodes. The results are gathered in Figure 38 and Table 8.

Analysing these results, it is clear that the agent performs better when the environment is simpler, and there is no network visualisation (albeit at the cost of a higher dispersion of the results). This was consistent with the results of the RSA Game 8, as displayed in Section 6.1.3, but with a higher performance delta. Moreover, it under performs the heuristics by around 10%, whilst having a higher standard deviation, even if in some

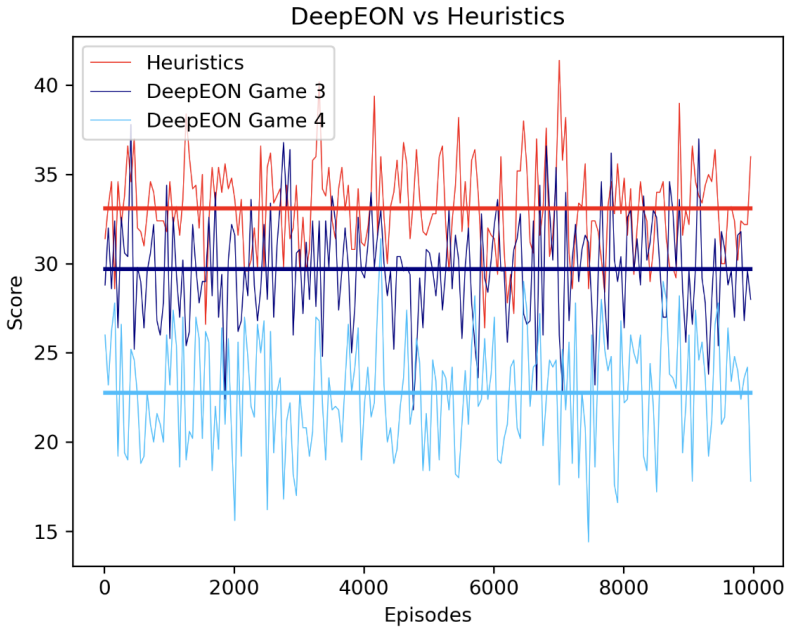


Figure 38: Final agent results (50 episode average)

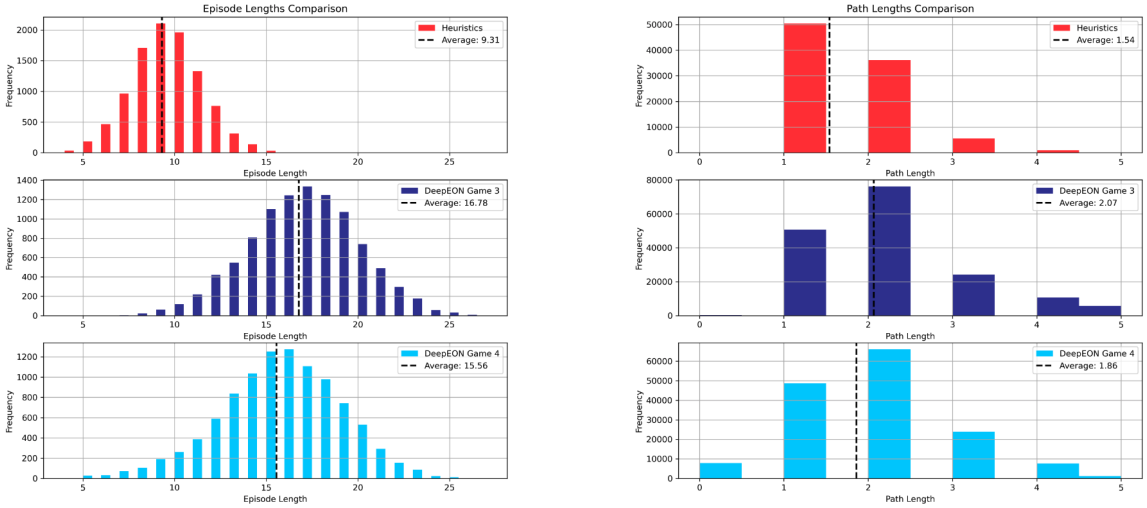
	Average Reward		Standard Deviation	
Model	Absolute	vs Heuristic	Absolute	vs Heuristic
Heuristics	33.10	NA	19.08	NA
Game 3	29.71	-10.24%	22.30	+17.81%
Game 4	22.77	-31.21%	20.86	+6.75%

Table 8: Agent performance versus heuristics

episodes it can be seen that the agent actually performs better than the heuristics. This seems interesting, as from Figures 35 and 36 there are some runs that actually perform better than the heuristics by a couple of points.

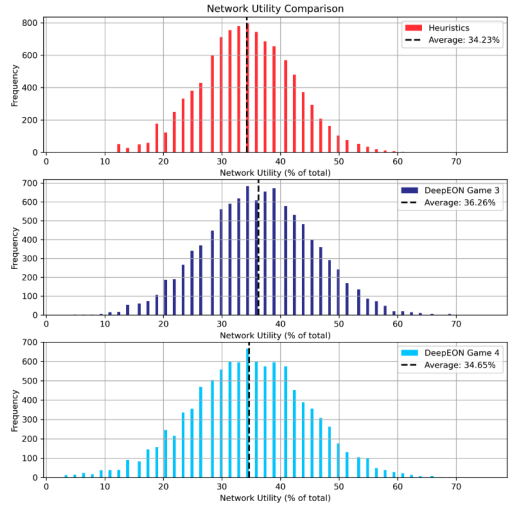
6.2.4 Analytics

Using the policy analysis tools developed, we obtained interesting results for the episode length, route length, network utility, and episode rewards for the models tested.

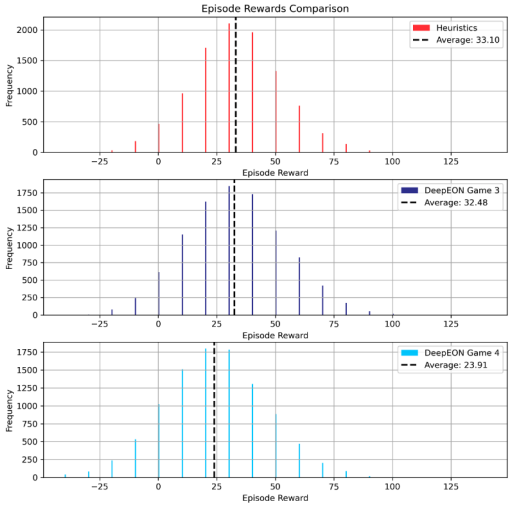


(a) Episode length distribution

(b) Route length distribution



(c) Network utility distribution



(d) Episode rewards distribution

Figure 39: Routing game analytics

As seen in Figure 39, the three models perform similar decisions at the macro level. In the case of episode rewards and network utilisation, all three share a very similar normal distribution but with slightly different mean values (as seen in Figures 39c and 39d). Figure 39b shows that the heuristics tends to have a shorter average route length than the DQN models (1.54 vs 2.07 and 1.86), with both models showing a similar distribution but having game 4 with a wider tail.

However, the more surprising graph where the three differ is in the episode length breakdown (Figure 39a). It can be seen that the heuristic usually has significantly shorter episodes than the models, with an average episode length of 9.31, compared to 16.78 and 15.56 for the two games. What this implies is that the heuristic is taking less actions, resulting in less links used for each route and thus, shorter routes. These shorter routes are most likely one of the main contributors to the increase in performance, suggesting the agent should be looking for more straightforward ways of moving from source to target,

avoiding moving through the network through long routes.

6.3 Overall Results

All in all, we did not manage to beat the First Fit heuristics at neither the RSA nor the Routing-only games, coming up with the following analysis of the results:

1. Spectrum allocation is significantly more challenging than routing for the agent. Both in the RSA and RSA non-routing setups, using both DQN and Dreamer models, the agent does not seem to get significantly closer to baseline heuristic performance, even after the game improvements.

2. Routing-only performance is better, but still fails to outperform the baseline. Even though, under some setups, the agent comes remarkably close to the heuristics (even outperforming it in some conditions), it still lags behind it. Analytics drawn from both seem to suggest that this is due to lower episode lengths (around 40%) for heuristic runs. However, the agent was better at allocating spectrum efficiently, with an average utility of 36.3% versus 34.2%.

7 Conclusions

This project aimed to explore how the use of Deep RL in arcade game representations of EONs could improve dynamic routing and spectrum allocation. Two different approaches were used: improving RSA game developed by M. Simon in [22], and developing a new routing-only game. As our baseline, we chose to compare the agent performance against a KSP-FF heuristic algorithm.

Three distinct games were created when improving RSA game, each with significant improvements and different visual representations. Two new environments and one new episode ending mechanism were introduced, along with a top-down view of a network, which allowed the DeepRL agent to see state of the entire network in each episode. Environment 2 resulted in mean increase of over 37% compared to environment 1. New episode ending mechanism did not result in a significant performance difference, and a lower number of slots resulted in better performance. Simplifying the game by removing routing problem improved the performance. Furthermore, adding a top-down network view did not result in significant differences in performance and the agent still failed to beat the heuristic. Additionally, a World Model DreamerV3 was explored, however, was still outperformed by the heuristic by over 20%.

The non-optimal performance could be could be due to a variety of factors: non-optimal game parameters and reward schemes, not enough training time or simply errors in game representations.

The routing game developed was built using a similar representation to the RSA game, but allowing the agent to fully focus on solving the routing problem, with a FF heuristic taking care of spectrum allocation. Two different action spaces were tested, arrows and teleport, with the teleport environment performing 14.4% higher than arrows during initial testing. Moreover, a game with a top-down view of the spectrum in the network was implemented, but performed 23.4% than the simpler game without it. Even though the routing game performed better than the combined RSA game, it still fell short from the heuristic by 10.2%.

In order to understand this better some analytics were drawn from both agent and heuristic episodes, suggesting that the main cause for the difference in the performance

was a significantly lower (around 40%) route length in heuristic runs. Nevertheless, the routing agent performed slightly better at filling up the network spectrum, with an average utility of 36.3% versus 34.2%.

This project shows that training an DeepRL agent using visual representation to beat KSP-FF heuristics is challenging, but it can be done under certain setups and conditions (like filling the spectrum more efficiently). Future work in the short-term should focus on building a more robust analytics platform to expand the explainability of the agent and fine-tuning the parameters further to increase the performance. Long-term, different game structures as well as models should be investigated to deliver a significant boost in performance.

8 Team Contributions

Overall, all team members had similar tasks and responsibilities throughout the development of the project, contributing to the theory, design, implementation and testing of the games.

The only significant difference in tasks was that the RSA game was mostly developed by Moshe Simon and Andrey Staniukynas, whereas the Routing game was primarily designed by Alina Stanoiu, Rafael Asensio and David Sal. However, as shown in this report, both groups frequently interacted with each other to share new findings, problem-solve, and push the project forward as a unified team.

A more granular breakdown of who worked on and reviewed each section of the report is shown below.

Moshe Simon (MS)

- *Written:* Technical Background of Your Work - Deep Reinforcement Learning (Section 4.3, Results and Discussion - RSA Game (Section 6.1).
- *Reviewed:* Literature Review (Section 2), Results and Discussion - Routing Game (Section 6.2).

Rafael Asensio (RA)

- *Written:* Project Goals (Section 3), Methods - Routing Game (Section 5.2), Results and Discussion - Routing Game and Overall (Sections 6.2 and 6.3), Conclusions (Section 7), Team Contributions (Section 8).
- *Reviewed:* Executive Summary, Methods - Upgrading the RSA Game (Section 5.1).

Andrey Staniukynas (AnS)

- *Written:* Methods - Upgrading the RSA Game (Section 5.1), Results and Discussion - RSA Game (Section 6.1), Conclusions (Section 7).
- *Reviewed:* Technical Background of Your Work (Section 4.3), Methods - Routing game (Section 5.2).

Alina Stanoiu (AlS)

- *Written:* Technical Background of Your Work - Optical Networks and Elastic Optical Networks (Sections 4.2 and 4.1), Methods - Routing Game (Section 5.2).
- *Reviewed:* Introduction (Section 1), Literature Review (Section 2), Results and Discussion (Section 6).

David Sal (DS)

- *Written:* Executive Summary, Introduction (Section 1), Literature Review (Section 2), Results and Discussion - Routing Game (Section 6.2).
- *Reviewed:* Technical Background of Your Work - Optical Networks and Elastic Optical Networks (Sections 4.2 and 4.1), Results and Discussion - RSA Game (Section 6.1).

References

[1] "Internet traffic statistics and trends in 2023," https://blog.gitnux.com/internet-traffic-statistics/, accessed: May 7, 2023.

[2] FS Community, "CWDM and DWDM ITU Channels Guide," https://community.fs.com/news/cwdmdwdm-itu-channels-guide.html, n.d., [Accessed: May 06, 2023].

[3] Y. Ujjwal and J. Thangaraj, "Review and analysis of elastic optical network and sliceable bandwidth variable transponder architecture," *Optical Engineering*, vol. 57, no. 11, p. 110802, 2018.

[4] R. Martinez, R. Casellas, R. Munoz, and T. Tsuritani, "Elastic optical networks: a new dawn for the optical layer?" *IEEE Communications Magazine*, vol. 50, no. 2, pp. s12–s20, February 2012.

[5] D. Sharma and S. Kumar, "An overview of elastic optical networks and its enabling technologies," *Int. J. Eng. Technol.(IJET)*, vol. 9, no. 3, pp. 1643–1649, 2017.

[6] A. Paul, Q. Rahman, S. Bandyopadhyay, and Y. P. Aneja, "A fast approach to solve the route and spectrum allocation problem in ofdm networks," *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*, pp. 92–98, 2017.

[7] J. W. Nevin, S. Nallaperuma, N. A. Shevchenko, Z. Shabka, G. Zervas, and S. J. Savory, "Techniques for applying reinforcement learning to routing and wavelength assignment problems in optical fiber communication networks," *Journal of Optical Communications and Networking*, vol. 1, 2022.

[8] Y. Yin, H. Zhang, M. Zhang, M. Xia, Z. Zhu, S. Dahlfort, and S. B. Yoo, "Spectral and spatial 2d fragmentation-aware routing and spectrum assignment algorithms in elastic optical networks," *Journal of Optical Communications and Networking*, vol. 5, no. 10, pp. A100–A106, 2013.

[9] Y. Sone, A. Hirano, A. Kadohata, M. Jinno, and O. Ishida, "Routing and spectrum assignment algorithm maximizes spectrum utilization in optical networks," in *2011 37th European Conference and Exhibition on Optical Communication*. IEEE, 2011, pp. 1–3.

[10] F. S. Abkenar and A. G. Rahbar, "Study and analysis of routing and spectrum allocation (rsa) and routing, modulation and spectrum allocation (rmsa) algorithms in elastic optical networks (eons)," *Optical Switching and Networking*, vol. 23, pp. 5–39, 2017.

[11] J. Suárez-Varela, A. Mestres, J. Yu, L. Kuang, H. Feng, A. Cabellos-Aparicio, and P. Barlet-Ros, "Routing in optical transport networks with deep reinforcement learning," *Journal of Optical Communications and Networking*, vol. 11, no. 11, pp. 547–558, 2019.

[12] P. Almasan, J. Suárez-Varela, B. Wu, S. Xiao, P. Barlet-Ros, and A. Cabellos-Aparicio, "Towards real-time routing optimization with deep reinforcement learning: Open challenges," in *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*. IEEE, 2021, pp. 1–6.

[13] A. B. Terki, J. Pedro, A. Eira, A. Napoli, and N. Sambo, "Routing and spectrum assignment assisted by reinforcement learning in multi-band optical networks," in *2022 European Conference on Optical Communication (ECOC)*, 2022, pp. 1–4.

[14] X. Chen, B. Li, R. Proietti, H. Lu, Z. Zhu, and S. B. Yoo, "Deeprmsa: a deep reinforcement learning framework for routing, modulation and spectrum assignment in elastic optical networks," *Journal of Lightwave Technology*, vol. 37, no. 16, pp. 4155–4163, 2019.

[15] X. Chen, R. Proietti, and S. B. Yoo, "Building autonomic elastic optical networks with deep reinforcement learning," *IEEE Communications Magazine*, vol. 57, no. 10, pp. 20–26, 2019.

[16] R. Weixer, S. Kühl, R. M. Morais, B. Spinnler, W. Schairer, B. Sommerkorn-Krombholz, and S. Pachnicke, "A reinforcement learning framework for parameter optimization in elastic optical networks," in *2020 European Conference on Optical Communications (ECOC)*. IEEE, 2020, pp. 1–4.

[17] J. Pinto-Ríos, F. Calderón, A. Leiva, G. Hermosilla, A. Beghelli, D. Bórquez-Paredes, A. Lozada, N. Jara, R. Olivares, and G. Saavedra, "Resource allocation in multicore elastic optical networks: A deep reinforcement learning approach," *arXiv preprint arXiv:2207.02074*, 2022.

[18] J. W. Nevin, S. Nallaperuma, N. A. Shevchenko, Z. Shabka, G. Zervas, and S. J. Savory, "Techniques for applying reinforcement learning to routing and wavelength assignment problems in optical fiber communication networks," *Journal of Optical Communications and Networking*, vol. 14, no. 9, pp. 733–748, 2022.

[19] B. Tang, Y.-C. Huang, Y. Xue, and W. Zhou, "Heuristic reward design for deep reinforcement learning-based routing, modulation and spectrum assignment of elastic

optical networks," *IEEE Communications Letters*, vol. 26, no. 11, pp. 2675–2679, 2022.

[20] J. M. Ziazet, "Reinforcement learning for routing, modulation and spectrum assignment problem in elastic optical networks," Ph.D. dissertation, Concordia University, Canada, 2019.

[21] T.Ray, T.Wei, T.Katerbau, and D.Alberg, "Artificial intelligence in elastic optical networks," Ph.D. dissertation, University College London, United Kingdom, 2022.

[22] S. Moshe, "Deep reinforcement learning applied in dynamic elastic optical networks using arcade-like game representation," Ph.D. dissertation, University College London, United Kingdom, 2022.

[23] M. T. P. W. B. Mukherjee, I. Tomkos and Y. Zhao, *Springer Handbook of Optical Networks*, ser. Springer Handbooks. Springer Cham, 2020.

[24] N. S. B. C. Chatterjee and E. Oki, "Routing and spectrum allocation in elastic optical networks: A tutorial," *IEEE Communications Surveys Tutorials*, vol. 17, no. 3, pp. 1776–1800, 2015.

[25] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," 2017.

[26] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap, "Mastering diverse domains through world models," *arXiv preprint arXiv:2301.04104*, 2023.

[27] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Dream to control: Learning behaviors by latent imagination," *arXiv preprint arXiv:1912.01603*, 2019.

Appendices

A Spectrum Allocation Data Structures

As mentioned earlier in the report, we created two novel data structures to keep track of spectrum allocation throughout the Routing game.

1.1 Link Grid

The link grid is used to store the spectrum allocated for the requested connections that have been established, and it is updated after a player reaches the target. It is used across multiple rounds (requests) and becomes empty at the start of each game (episode). The structure of the link grid is shown in Figure 40. A dictionary stores arrays that represent the spectrum for each link using the nodes in between which the link is formed as the access key: `NODE1-NODE2`. Each position in the array represents a spectrum slot and it will take a value of 1 (unavailable) or 0 (available).

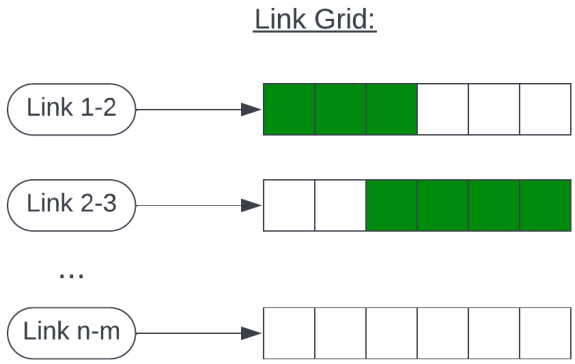


Figure 40: Link Grid structure

1.2 Running Path Spectrum (RPS)

The RPS is used to keep track of the spectrum availability during one request. A new RPS is initialized at the beginning of each request. In order for a spectrum to be allocated at the end of the request, the slots have to meet the continuity and contiguity constraint across all links in the route and to comply with the First Fit allocation.

From a structural point of view, RPS is a dictionary that stores one matrix for each link, using the same key structure as link grid. (Figure 41) At the begging of a round, each matrix provides all the possible options for the spectrum allocation of the current request without taking in consideration any availability.

When checking the availability of a link, the RPS will be updated by deleting the options that are not available. First, the options in the matrix are compared with the link grid. The second step is to compare each option in the rps of the current link with all the options in all the links that have been selected across the route. A link will not be visited twice in a route. The options in the matrix are in order, so when the target was reached, the first available option in the RPS of the last link of the route is selected.

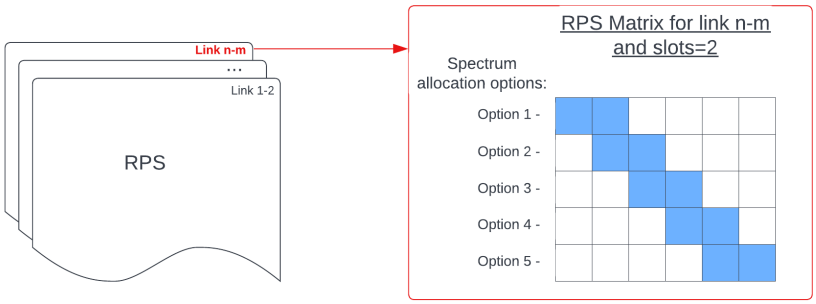


Figure 41: RPS structure

B Agent Game Implementation

The diagram in Figure 42 presents the overall implementation of the agent game and its integration with a custom environment. This section will cover only the game implementation.

The `init()` method is used to set game parameters that are used across all played games such as the network topology, size of the game window, and background color, and also to call a new game. The `new_game()` method creates a new game, by defining a counter for the number of blocks and an empty link grid. Then, from here the `new_round()` method is called which generates a new request, updates the position of the player/agent and initialises the variables that are needed for the round.

A key method for the functionality of the game is `check_solution()`. In the case of the agent game, this provides interaction with the custom environment and enables progression during the game. The method receives a node as input. If the node is available from the current node, then it will enable the movement to that node by calling method `move_to_node()`. Then, if the provided node is the target a new round is called and the reward and link grid are updated. Otherwise, the round continues. If an unavailable node is provided, the number of blocks counter is increased and the corresponding reward is updated.

When method `move_to_node()` is called, a series of changes in the game are triggered which are responsible for implementing the spectrum allocation. The constructed route and the node availability are updated. When updating the node availability, the methods `check_with_link_grid()` and `check_with_history()` implement the changes in the RPS and array `nodes_availability` stores the state of the nodes after those computations in order to make it more accessible throughout the code.

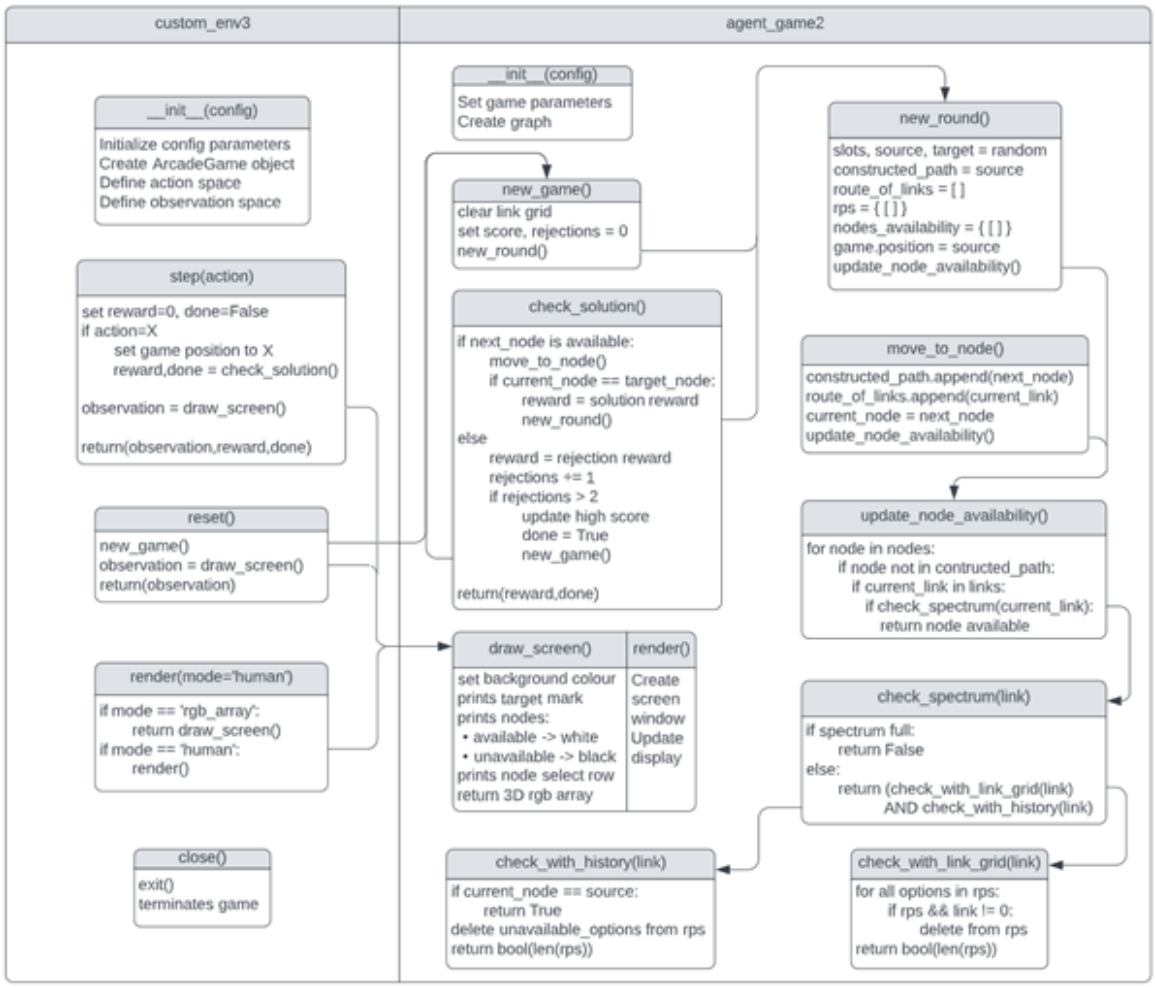


Figure 42: Architecture of the agent game and interaction with custom environment