Deep Reinforcement Learning meets Graph Neural Networks: exploring a routing optimization use case

Preprint · October 2019

DOI: 10.48550/arXiv.1910.07421

6 authors, including:

Paul Almasan
Telefónica I+D
29 PUBLICATIONS **870** CITATIONS
SEE PROFILE

Jose Suarez-Varela
Telefónica I+D
58 PUBLICATIONS **1,716** CITATIONS
SEE PROFILE

Krzysztof Rusek
AGH University of Krakow
65 PUBLICATIONS **1,238** CITATIONS
SEE PROFILE

Pere Barlet-Ros
Polytechnic University of Catalonia
167 PUBLICATIONS **3,472** CITATIONS
SEE PROFILE

Deep Reinforcement Learning meets Graph Neural Networks: exploring a routing optimization use case

Paul Almasan
Barcelona Neural Networking Center
Universitat Politècnica de Catalunya

José Suárez-Varela
Barcelona Neural Networking Center
Universitat Politècnica de Catalunya

Arnau Badia-Sampera
Barcelona Neural Networking Center
Universitat Politècnica de Catalunya

Krzysztof Rusek
Barcelona Neural Networking Center, Universitat Politècnica de Catalunya
AGH University of Science and Technology

Pere Barlet-Ros
Barcelona Neural Networking Center
Universitat Politècnica de Catalunya

Albert Cabellos-Aparicio
Barcelona Neural Networking Center
Universitat Politècnica de Catalunya

ABSTRACT

Recent advances in Deep Reinforcement Learning (DRL) has shown a dramatic improvement in decision-making problems. As a result, DRL seems promising to solve many relevant network optimization problems (e.g., routing) in self-driving software-defined networks. However, most state-of-the-art DRL-based networking techniques fail to *generalize*, which means that they perform well in network topologies seen during training, but not over new topologies. The reason behind this important limitation is that existing DRL networking solutions use standard neural networks (e.g., fully connected, convolutional) that are not suited to learn graph-structured information. In this paper we propose to use Graph Neural Networks (GNN) in combination with DRL. GNN have been recently proposed to model graphs, and our novel DRL+GNN architecture is able to *learn and generalize over arbitrary network topologies*. To showcase its generalization capabilities, we evaluate it on a SDN-based Optical Transport Network (OTN) scenario, where traffic demands need to be allocated efficiently. Our results show that our DQN+GNN agent is able to achieve outstanding performance in topologies never seen during training.

1 INTRODUCTION

Recent advances in Deep Reinforcement Learning (DRL) showed a significant improvement in decision-making and automated control problems [21, 24]. In this context, the network community is investigating DRL as a key technology for network optimization (e.g., routing) with the goal of enabling self-driving software-defined networks [19]. However, existing DRL-based solutions still fail to *generalize* when applied to different network scenarios. This hampers the ability of the DRL agent to make good decisions when facing a network state not seen during training.

Indeed, most of existing DRL proposals for networking can only operate over the same network topology seen during training [8, 17] and, thus, cannot generalize and efficiently operate over unseen network topologies. The main reason behind this strong limitation is that computer networks are fundamentally represented as graphs. For instance, the network topology and routing policy are typically represented as such. However, state-of-the-art proposals [8, 18, 27, 28] use traditional neural network (NN) architectures (e.g., fully-connected, Convolutional Neural Networks) that are not well suited to model graph-structured information [4].

Recently, Graph Neural Networks (GNN) [23] have been proposed to model and operate over graphs with the aim of achieving relational reasoning and combinatorial generalization. In other words, GNNs facilitate learning the relations between graph elements and the rules for composing them. GNNs have shown unprecedented generalization capabilities in the field of networking modeling and optimization [22, 26].

In this paper, we present a pioneering DRL+GNN architecture for network optimization. Particularly, our novel architecture is intended to optimize network routing and to be able to generalize over never-seen arbitrary topologies. The GNN integrated in our DRL agent is inspired by Message-passing Neural Networks [12], which were successfully applied to solve a relevant chemistry-related problem. In our case, the GNN was specifically designed to capture meaningful information about the relations between the links and the traffic flowing through the network topologies.

To showcase the generalization capabilities of the proposed DRL+GNN architecture, we experimentally evaluate it

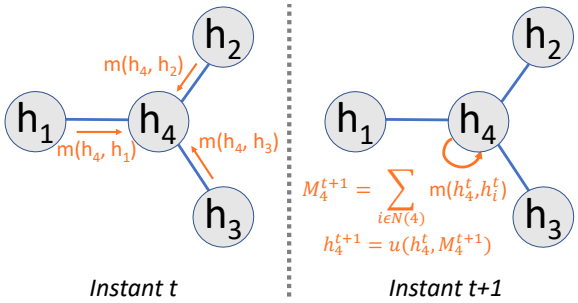


Figure 1: Message passing overview for node 4 in two time steps.

in a SDN-based Optical Transport Network (OTN) scenario. In particular, the DRL+GNN agent is tasked to allocate traffic demands as they arrive, maximizing the traffic volume routed through the network. The evaluation results show that our agent achieves a strong generalization capability compared to state-of-the-art DRL (SoA DRL) algorithms. Particularly, we trained both DRL-based solutions – the DRL+GNN and a SoA DRL agent – in a network topology and evaluated them in a second network topology. The results show that our DRL+GNN agent clearly outperforms the SoA DRL agent in the new topology. Additionally, to further test the generalization capability of the proposed DRL-based architecture, we evaluated it in a set with 136 different real-world network topologies. The results show that our agent is able to achieve outstanding performance over the networks never seen during training.

To the best of our knowledge, this is the first DRL-based proposal that integrates a GNN to achieve generalization in network optimization problems.

2 BACKGROUND

The solution proposed in this paper combines two machine learning mechanisms. First, we use a Graph Neural Network to model computer network scenarios. Second, we use Deep Reinforcement Learning to build an agent that learns how to efficiently operate networks following a particular optimization goal.

2.1 Graph Neural Networks

Graph Neural Networks are a novel family of neural networks designed to operate over graph-structured information. They were introduced in [23] and numerous variants have been developed since [10, 16, 30]. In their basic form, they consist in associating some initial states to the different elements of an input graph, and combine them considering how these elements are connected in the graph. An iterative message-passing algorithm updates the elements' state and uses the resulting states to produce an output. The particularities of the problem to solve will determine which GNN variant is more suitable, depending for instance, on the nature of the graph elements (i.e., nodes and edges) involved.

Message Passing Neural Networks (MPNN) [12] are a well-known type of GNNs that apply an iterative message-passing algorithm to propagate information between the nodes of the graph. In a message-passing step (see Figure 1), each node k receives messages from all the nodes in its neighborhood, denoted by $N(k)$. Messages are generated by applying a message function $m(\cdot)$ to the hidden states of node pairs in the graph, and then are combined by an aggregation function, for instance, a sum (Equation 1). Finally, an update function $u(\cdot)$ is used to compute a new hidden state for every node (Equation 2).

$$M_k^{t+1} = \sum_{i \in N(k)} m(h_k^t, h_i^t) \tag{1}$$

$$h_k^{t+1} = u(h_k^t, M_k^{t+1}) \tag{2}$$

Where functions $m(\cdot)$ and $u(\cdot)$ can be learned by neural networks. After a certain number of iterations, the final node states are used by a readout function $r(\cdot)$ to produce an output for the given task. This function can also be implemented by a neural network and is typically tasked to predict properties of individual nodes (e.g., the node's class) or global properties of the graph.

GNNs have been able to achieve relevant performance results in multiple domains where data is typically structured as a graph [3, 12]. Since computer networks are fundamentally represented as graphs, it is inherent in their design that GNNs offer unique advantages for network modeling compared to traditional neural network architectures (e.g., fully connected NN, convolutional NN, etc.). Recent work has shown their potential for network performance prediction [22], even when making predictions on network topologies unseen during training [26].

2.2 Deep Reinforcement Learning

DRL algorithms aim at learning a long-term strategy that leads to maximize a goal function in an optimization problem. DRL agents start from *tabula rasa*. This means that they have no previous expert knowledge about the environment where they operate. They learn the optimal strategy by an iterative process that explores the state and action spaces and is directed by a reward function. The state and action spaces are denoted by a set of states ($\mathcal{S}$) and a set of actions ($\mathcal{A}$). Given a state $s \in \mathcal{S}$, the agent will perform an action $a \in \mathcal{A}$ that produces a transition to a new state $s' \in \mathcal{S}$, and will provide the agent with a reward r. Then, the objective is to find a strategy that maximizes the cumulative reward at the

end of an episode. The definition of the end of an episode depends on the optimization problem to address. This optimization problem can be modeled as a Markov Decision Process (MDP). However, finding the optimal solution to the MDP requires to evaluate all the possible combinations of state-action pairs.

An alternative to solve the MDP is using Reinforcement Learning (RL). Q-learning [31] is a RL algorithm whose goal is to make an agent learn a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$. The algorithm creates a table (also known as q-table) with all the possible combinations of states and actions. At the beginning of the training, the table is initialized (e.g., with zeros or random values) and during training, the agent updates these values according to the rewards obtained after selecting an action. These values, called q-values, represent the expected cumulative reward after applying action *a* from state *s*, assuming that the agent follows the current policy π learned during the rest of the episode. During training, q-values are updated using the Bellman equation (see Equation 3) where *r(s,a)* is the reward obtained from selecting action *a* from state *s*, *Q(s',a)* is the q-value function and $\gamma \in [0, 1]$ is the discount factor, which represents the importance of the rewards obtained in the future:

$$Q(s, a) = r(s, a) + \gamma \max_{a} Q(s', a) \tag{3}$$

Deep Q-Network (DQN) [20] is a more advanced algorithm based on Q-learning that uses a Deep Neural Network (DNN) to approximate the q-value function. As the q-table size becomes larger, Q-learning has difficulties to learn a policy from high dimensional state and action spaces. To overcome this problem, they proposed to use a DNN as q-value function estimator. This enables to rely on the generalization capabilities of DNNs to estimate the q-values of states and actions unseen in advance. For this reason, a NN well suited to understand and generalize over the input data of the DRL agent is crucial for the agents to perform well when facing states (or environments) never seen before. Additionally, DQN uses an experience replay buffer to store past sequential experiences (i.e., stores tuples of *{s,a,r,s'}*). While training the neural network, the temporal correlation is broken by sampling randomly from the experience replay buffer.

3 NETWORK OPTIMIZATION SCENARIO

Finding the optimal routing configuration for a set of traffic demands is a NP-hard problem [11]. This makes it necessary to explore alternative solutions (e.g., heuristics) with the aim of maximizing the performance at an affordable cost. In this paper, we explore the potential of a GNN-based DRL agent to operate and generalize over routing scenarios involving diverse network topologies. As a first approach, we consider a routing scenario in Optical Transport Networks (OTN)

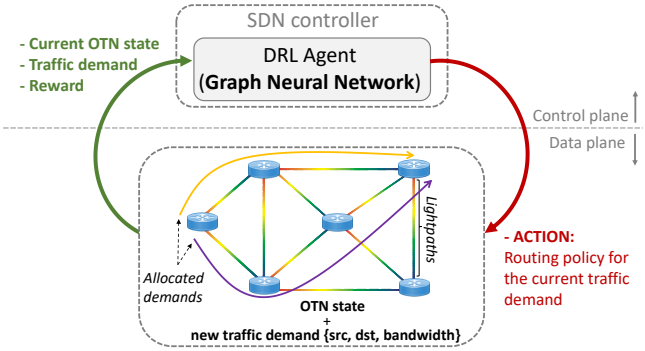


Figure 2: Schematic representation of the DRL agent in the SDN-based OTN routing scenario.

[25]. Particularly, we consider a SDN-based scenario where the DRL agent (located in the control plane) has a global view of the current network state, and has to make routing decisions on every traffic demand as it arrives. This is a relevant optimization scenario that has been studied in the last decades in the optical networking community, where many solutions have been proposed [8, 15, 28].

This problem can be described as a classical resource allocation problem in networks. In our OTN scenario, we have a topology where links have a given capacity, and the DRL agent receives traffic requests with different bandwidth requirements that need to be allocated in real time. Once a traffic demand is allocated, the resources for this demand cannot be freed or replaced. Thus, the problem consists of allocating the maximum traffic volume in the network.

The DRL agent operates at the level of the electrical domain, over a logical topology where the nodes represent Reconfigurable Optical Add-Drop Multiplexer (ROADM) nodes and edges some predefined lightpaths connecting them (see Figure 2). The DRL agent receives a traffic demand defined by the tuple $\{src, dst, bandwidth\}$, and is tasked to route the traffic demand through a particular sequence of lightpaths (i.e., an end-to-end path) that connect the source and the destination of such demand. In Figure 2, we can see a graphical representation of the network state after allocating two demands. Traffic demands are considered requests of Optical Data Units (ODU) whose bandwidth demands are defined in the ITU-T Recommendation G.709 [1].

In this scenario, the routing problem is defined as finding the optimal routing policy for each incoming source-destination traffic demand. The learning process is guided by an objective function that aims to maximize the traffic volume allocated in the network in the long-term. We consider that a demand is properly allocated if there is enough available capacity in all the lightpaths forming the end-to-end path selected. Note that lightpaths are the links/edges in the logical topology. The demands do not expire, occupying

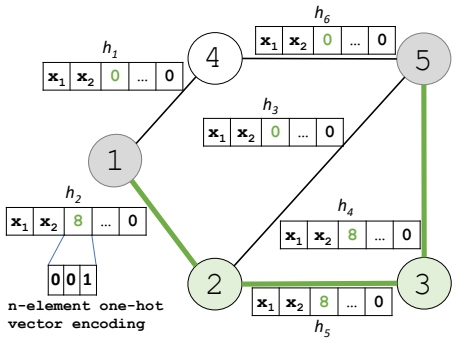


Figure 3: Action representation in the link hidden states.

the lightpaths until the end of an episode. This implies a challenging task for the DRL agent, since it has not only to identify critical resources on networks (e.g., potential bottlenecks) but also to deal with the uncertainty in the generation of future traffic demands.

4 GNN-BASED DRL AGENT DESIGN

In this section, we describe the DRL+GNN agent proposed in this paper. Our DRL agent implements the DQN algorithm [20], where the q-value function is modeled by a GNN. From now on we refer to our agent as DQN+GNN agent. At each time step, the agent receives a graph-structured network state observation and a traffic demand as inputs. Particularly, the network state includes the topology with some link-level features (e.g., utilization). Then, the objective of the GNN is to estimate the q-value $Q(s, a)$ of applying a routing action (a) for the new traffic demand from the current network state (s).

In our case, the GNN constructs a graph representation where the links of the topology are the graph entities. In this representation, the link hidden states are initialized considering the input link-level features and the routing action to evaluate (see more details in subsections 4.1, 4.2 and 4.3). With this representation, an iterative message passing process runs between the link hidden states according to the graph structure.

At the end of the message passing phase, the GNN outputs a q-value estimate. This q-value is evaluated over a limited set of actions ($\mathcal{A}$), and finally the DRL agent selects the action with higher q-value. The selected action is then applied over the environment, producing a transition to a new network state. If the action performed was successful (i.e., all the links selected had enough available capacity to support the new demand), a positive reward is returned to the agent, otherwise the episode ends and the DRL agent receives a reward equal to zero.

Table 1: Input features of the link hidden states

Notation	Description
x_1	Link available capacity
x_2	Link Betweenness
x_3	Action vector (bandwidth allocated)
$x_4 - x_N$	Zero padding

During the training phase, an *ϵ-greedy* exploration strategy [20] is used to select the next action applied by the agent. This means that a random action is executed with probability ϵ, while the action with higher q-value is selected with probability $(1 - \epsilon)$.

4.1 Environment

The network state is defined by the topology links' features, which include the link capacity and link betweenness. The former indicates the amount of capacity available on the link. The latter is a measure of centrality inherited from graph theory that indicates how many paths may potentially traverse the link. In particular, we compute the link betweenness in the following way: for each pair of nodes in the topology, we compute k candidate paths (e.g., the k shortest paths) , and we maintain a per-link counter that indicates how many paths pass through the link. Thus, the betweenness on each link is the number of end-to-end paths crossing the link divided by the total number of paths.

4.2 Action set

In this section we describe how the routing actions are represented in the DQN+GNN agent. Note that the number of possible routing combinations for each source-destination node pair typically results in a high dimensional action space in large-scale real-world networks. This makes the routing problem complex for the DRL agent, since it should estimate the q-values for all the possible actions (i.e., routing configurations) to apply. To overcome this problem, the action set must be carefully designed to reduce the dimensionality. Also, to leverage the generalization capability of GNN, the action should be introduced into the agent in the form of a graph. This makes the action representation invariant to node and edge permutation, which means that, once the GNN is successfully trained, it should be able to understand actions over arbitrary graph structures (i.e., over different network states and topologies).

Considering the above, we limit the action set to k candidate paths for each source-destination pair. To maintain a good trade-off between flexibility to route traffic and the cost to evaluate all the possible actions, we selected a set with

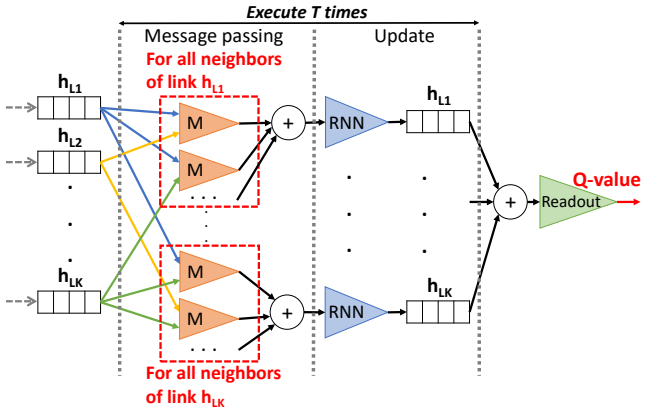


Figure 4: Message passing architecture.

Algorithm 1 Message Passing

Input : $\mathbf{x}_l$
Output : $\mathbf{h}_l^T, q$
1: **for each** $l \in \mathcal{L}$ **do**
2: $\quad h_l^0 \leftarrow [\mathbf{x}_l, 0 \ldots, 0]$
3: **for** $t = 1$ to T **do**
4: $\quad$ **for each** $l \in \mathcal{L}$ **do**
5: $\quad\quad M_l^{t+1} = \sum_{i \in N(l)} m\left(h_l^t, h_i^t\right)$
6: $\quad\quad h_l^{t+1} = u\left(h_l^t, M_l^{t+1}\right)$
7: $r \leftarrow \sum_{l \in \mathcal{L}} h_l$
8: $q \leftarrow R(r)$

the *k=4* shortest paths (by number of hops) for each source-destination node pair. This follows a criteria originally proposed by [28]. Note that the action set differs depending on the source and destination nodes of the traffic demand to be routed.

To represent the action, we introduce it within the network state. Particularly, we consider an additional link-level feature, which is the bandwidth allocated over the link after applying the routing action. This value corresponds to the bandwidth demand of the current traffic request to be allocated. Likewise, the links that are not included in the path selected by the action will have this feature set to zero. Since our OTN environment has a limited number of traffic requests with various discrete bandwidth demands, we represent the bandwidth allocated with a n-element one-hot encoding vector. Figure 3 illustrates the representation of this feature in the hidden state of the links in a simple network scenario. A traffic request from node 1 to node 5, with a traffic demand of 8 bandwidth units, is allocated over the path formed by the nodes $\{1, 2, 3, 5\}$. To summarize, Table 1 provides a description of the features included in the links' hidden states. These values represent both the network state and the action, which is the input needed to model the q-value function $Q(s, a)$. Note that the size of the hidden states is typically larger that the number of features in the hidden states. Then, a common approach is to fill the vector with zero values.

4.3 GNN architecture

The GNN model is based on the Message Passing Neural Network [12] model. In our case, we consider the link entity and perform the message passing process between all links. The algorithm performs T steps of message passing. A graphical representation can be seen in Figure 4. At the beginning, the algorithm iterates over all links of the network topology. For each link, it iterates over all its neighbors, combining the link features with a fully-connected NN. The outputs of these operations are called *messages* according to the GNN notation. Afterwards, the *messages* computed for a same node with its neighbors are aggregated using an element-wise sum. Lastly, a Recurrent NN (RNN) is used to update the link hidden states h_{LK} with the new aggregated information [i,e, $RNN(h_{LK}^t, h_{LK}^{t-1})$]. This process is repeated T times and, at the end of this phase, the resulting link states are aggregated using an element-wise sum. Finally, the result is passed through a fully-connected NN, which models the Readout function of the GNN. The output of this latter function is the estimated q-value of the input state and action.

Algorithm 1 shows a formal description the message passing process. The algorithm receives as input the links' features (x_l) and outputs a q-value (q). The loop from line 1 initializes the links' hidden states with the links' features. Afterwards, the algorithm performs the message passing during T steps (lines 3-6). In more detail, for each link l the message function $m(\cdot)$ described in Equation 1 takes as input the hidden state of a link h_l and the hidden state of a neighbor link h_i where $i \in N(l)$. The same process is repeated for all the neighbors of the link l and executed over all the links in the graph ($\mathcal{L}$). After iterating over all links, the output vectors are aggregated using an element-wise sum, resulting on a new feature vector for each link. The resulting vectors M_l^{t+1} are then combined with the respective previous link hidden states h_l^t by applying the update function $u(\cdot)$. Both the $m(\cdot)$ and $u(\cdot)$ functions are learned by neural networks (fully-connected and recurrent NN respectively). Finally, the outputs of the update function $u(\cdot)$ are aggregated using an element-wise sum (line 7), passing the result to a fully-connected neural network that models the readout R(·). This will output a numerical value that represents the q-value of the input state and action.

Algorithm 2 DRL Agent operation

```
1:  s, src, dst, bw ← env.init_env()
2:  reward ← 0
3:  k ← 4
4:  agt.mem ← {}
5:  Done ← False
6:  while not Done do
7:      k_q_values ← {}
8:      k_shortest_paths ← compute_k_paths(k, src, dst)
9:      for i in 0, ..., k do
10:         p' ← get_path(i, k_shortest_paths)
11:         s' ← env.alloc_demand(s, p', src, dst, dem)
12:         k_q_values[i] ← compute_q_value(s', p')
13:     q_value ← epsilon_greedy(k_q_values, ϵ)
14:     a ← get_action(q_value, k_shortest_paths, s)
15:     r, Done, s', src', dst', bw' ← env.step(s, a)
16:     agt.rmb(s, src, dst, bw, a, r, s', src', dst', bw')
17:     reward ← reward + r
18:     If training_steps % M == 0: agt.replay()
19:     src ← src'; dst ← dst'; bw ← bw', s ← s'
```

4.4 DRL Agent Operation

The DRL agent operates by interacting with the environment. In Algorithm 2 we can observe a pseudo-code describing the DRL agent operation. At the beginning, we initialize the environment s by setting all the link capacities to the maximum value and computing the link betweenness. At the same time, the environment generates a traffic demand to be allocated defined by the tuple $\{src, dst, bw\}$. We also initialize the cumulative reward to zero, define the action set size and create the experience replay buffer (*agt.mem*). Afterwards, we execute a while loop (lines 6-19) that finishes when there is some demand that cannot be allocated in the network topology (i.e., there is at least one link in the selected path that does not have enough capacity). The *k=4* shortest paths are computed in line 8. For each path, we allocate the demand along all the links forming the path and compute a q-value (lines 10-12). Once we have the q-value for each state-action pair, the next action a to apply is selected using an ϵ-greedy exploration strategy (line 13). The action (i.e., allocate on the *k-th* shortest path) is then applied to the environment, leading to a new state *s'*, a reward r and a flag *Done* indicating if there is some link without enough capacity to support the demand. Additionally, the environment returns a new traffic demand tuple $\{src', dst', bw'\}$. The information about the state transition is stored in the experience replay buffer (line 16). This information will later be used to train the GNN in the *agt.replay()* call (line 18), which is executed every M training iterations.

5 EXPERIMENTAL RESULTS

In this section we train and evaluate our GNN-based DRL agent to efficiently allocate traffic demands in the SDN-based OTN routing scenario described in Section 3. Particularly, the experiments performed in this section are focused on evaluating the generalization capabilities of our DQN+GNN agent (Section 4).

5.1 Evaluation setup

We implemented the DQN+GNN agent using Tensorflow. The DRL network topology environment was implemented using the OpenAI Gym framework [7]. The source code, together with all the training and evaluation results presented in this paper, are publicly available[1].

We trained the DQN+GNN agent on a single topology. The topology used was the 14-node NSFNet topology [13] and the agent was trained during 1,000 training episodes. For each iteration, we execute 100 and 50 training and testing episodes respectively. The model with highest testing score was picked to further evaluate on different topologies and compare with state-of-the-art solutions.

We consider 3 types of traffic demands whose bandwidth requirements are expressed in terms of multiples of ODU0 signals (i.e., 8, 32 and 64 ODU0 bandwidth units respectively) [1]. When the DRL agent allocates a demand, it receives an immediate reward being the bandwidth of the current traffic demand if it was properly allocated, otherwise the reward is 0. Traffic demands are generated on every step by uniformly selecting a source-destination node pair and a traffic demand bandwidth. This represents a challenging optimization scenario for the DQN+GNN agent, since the generation of new demands is random. Thus, it cannot exploit useful information from the traffic distribution to devise a proper long-term strategy.

Preliminary experiments were carried to choose an appropriate gradient-based optimization algorithm and hyperparameter values for our DQN+GNN agent. In our experiments, we select a size of 25 for the links' hidden states h_l. Note that the size of the hidden states is related to the amount of information they may potentially encode. For larger network topologies, it might be necessary to use larger sizes for the hidden state vectors. In every forward propagation we execute *T=8* message passing steps using batches of 32 samples. The optimizer used is a Stochastic Gradient Descent [5] method with Nesterov Momentum [29]. Regarding the hyperparameters, we use a learning rate of 10^{-4}, and a momentum of 0.9. For the ϵ-greedy exploration strategy, we start with ϵ=1.0 and this value is maintained during 10 training iterations. Afterwards, ϵ decays exponentially every 2 episodes. To stabilize the learning process, we re-train

[1] https://github.com/knowledgedefinednetworking/DRL-GNN

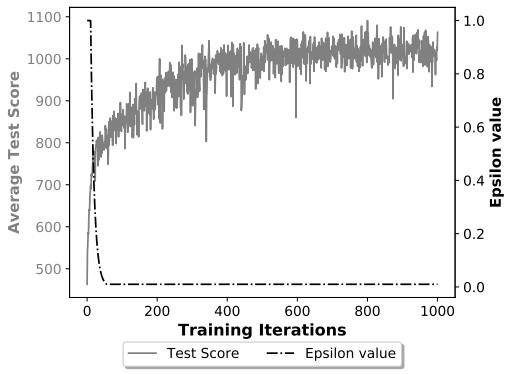


Figure 5: Average score evolution during training.

the weights of the GNN model every 2 episodes and every time we use 5 batches (with 32 samples) from the experience buffer. The experience replay buffer size was set to store 5,000 samples and it is implemented as a FIFO queue: once it is full, the oldest experience is removed.

5.2 Methodology

To evaluate the performance of our DQN+GNN agent we designed a wide range of evaluation experiments. In the first experiment, we aim to compare the performance and generalization capability of our DQN+GNN agent with respect to state-of-the-art DRL-based solutions. To do that, we take as reference the DRL-based solution proposed in [28]. We evaluate both DRL-based solutions in a network topology not seen during training. In a second block of experiments , we evaluate the performance and generalization capabilities of the proposed DRL+GNN architecture over real-world network topologies extracted from the Internet Topology Zoo dataset [14].

We compare our agent with a state-of-the-art DRL-based solution [28], a load balancing routing policy (LB) and a theoretical fluid model (labeled as *Theoretical Fluid*). The LB policy selects randomly one path among the *k=4* candidate shortest paths. The fluid model is a theoretical approach which considers that traffic demands may be split into the *k=4* candidate paths proportionally to the available capacity they have. This routing policy is aimed at avoiding congestion on links. For instance, paths with low available capacity will carry a small proportion of the traffic volume from new demands. Note that this model is non realizable because ODU demands cannot be split in real OTN scenarios. However, we use it as a reference to measure the performance of our DRL agent in different network scenarios.

5.3 Training

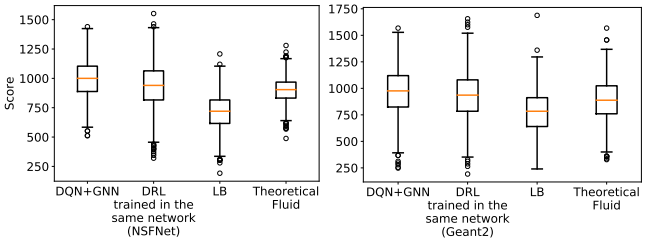
We train the DRL agent on an OTN routing scenario in the 14-node NSFNet topology [13], where we consider that the links represent lightpaths with capacity for 200 ODU0 signals. Note that the capacity is shared on both directions of the links and that the bandwidth of different traffic demands is expressed in multiples of ODU0 signals (i.e., 8, 32 or 64 ODU0 bandwidth units). During training, the agent receives traffic demands and allocates them on one of the *k=4* shortest paths available in the action set. We run 1,000 training iterations, storing the state transitions, actions performed and the rewards obtained in the experience replay buffer. We train the GNN by sampling from the experience buffer and use as output labels the q-values (defined in Equation 3). For the evaluation, we run 50 episodes and compute the average cumulative reward obtained over all of them.

In Figure 5, we show the average evaluation score of the DQN+GNN agent over the 50 episodes. We can observe that the learning remains stable after 600 training iterations approximately. We also show the evolution of the ϵ parameter used for the ϵ-greedy exploration strategy during the training. As we can observe, when ϵ starts to decay (i.e., around iteration 10) there is a constant and stable increase in the score achieved by the agent. This suggests that, at this point, the GNN is already able to produce sufficiently good q-value estimates to enable a smarter exploration of states and actions, which helps the agent to accelerate the learning process.

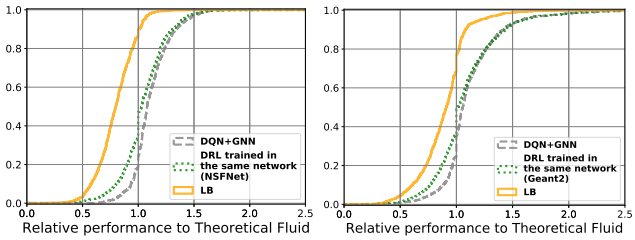
5.4 Evaluation against state-of-the-art DRL-based solutions

For this evaluation experiment, we selected the DQN+GNN agent that achieved highest score during training. We compare it against state-of-the-art DRL-based solutions. Particularly, we adapted the solution proposed in [28] to operate in scenarios where links share their capacity in both directions. We train two different instances of the state-of-the-art DRL agent in two network scenarios: the 14-node NSFNet and the 24-node Geant2 topology [2]. In our evaluation, we make 1,000 experiments with random traffic generation to provide representative results. Note that both, the DQN+GNN agent proposed in this paper and the state-of-the-art DRL solution, are evaluated over the same list of randomly generated demands.

We run two experiments to compare the *performance* of our DQN+GNN with the results obtained by the state-of-the-art DRL. In the first experiment, we evaluate the DQN+GNN agent against the state-of-the-art DRL agent trained on NSFNet, the LB routing policy and the theoretical fluid model. We evaluate the four routing strategies on the NSFNet topology and compare their performance. In Figure 6a we can observe a bloxplot with the evaluation results over 1,000 evaluation experiments, and Figure 6c shows the cumulative distribution function (CDF) of the relative score

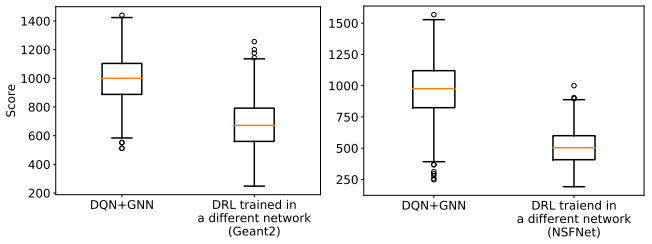


(a) Evaluation on NSFNet (b) Evaluation on Geant2

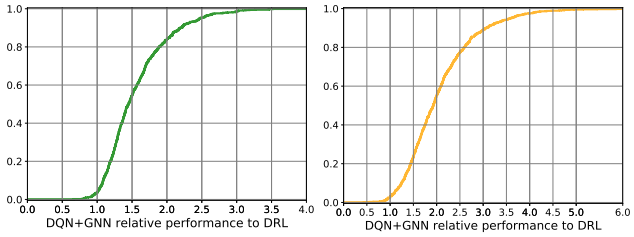


(c) Evaluation on NSFNet (d) Evaluation on Geant2

Figure 6: Performance evaluation against state-of-the-art DRL



(a) Evaluation on NSFNet (b) Evaluation on Geant2



(c) Evaluation on NSFNet (d) Evaluation on Geant2

Figure 7: Evaluation of the generalization capability of DRL-based solutions

obtained with respect to the fluid model. From these results, we can state that our DQN+GNN agent outperforms the state-of-the-art DRL-based solution in around 80% of the experiments. In the second experiment, we evaluate the same models (DQN+GNN, LB and Theoretical Fluid), but in this case the state-of-the-art DRL agent trained on Geant2. The resulting boxplot can be seen in Figure 6b and the CDF of the evaluation samples in 6d. Similarly, the performance of our agent is better in 60% of the cases. Thus, we can state our DQN+GNN agent has better performance than traditional DRL-based solutions.

We run two other experiments to compare the *generalization capabilities* of our DQN+GNN agent. In the first experiment, we evaluate our DQN+GNN agent (trained on NSFNet) against the state-of-the-art DRL agent trained on Geant2 and evaluate them on the NSFNet topology. In Figure 7a we can observe a boxplot with the evaluation results over 1,000 evaluation experiments. Figure 7c shows the CDF of the relative score of the proposed agent with respect to the state-of-the-art DRL agent trained on Geant2. From these results, we can observe that our DQN+GNN agent clearly outperforms the state-of-the-art DRL agent. For instance, in around 80% of the experiments the performance improvement is above 20%. In the second experiment, we evaluate the same DQN+GNN agent (trained on NSFNet) against the state-of-the-art DRL agent trained on NSFNet and evaluate both agents on the Geant2 topology. The resulting boxplot can be seen in Figure 7b and the corresponding CDF in Figure 7d. The results indicate that in this scenario our agent also outperforms the state-of-the-art DRL agent. In this case, in 80% of the experiments our DRL agent achieved more than 45% of performance improvement with respect to the state-of-the-art proposal. These results show that while the proposed DQN+GNN agent is able to generalize and achieve outstanding performance in the unseen Geant2 topology (Figures 7b and 7d), the state-of-the-art DRL agent performs poorly when applied to topologies not seen during training. This reveals the lack of generalization capability of the latter DRL-based solution compared to the agent proposed in this paper.

5.5 Generalization over other real-world network topologies

In this section we evaluate the generalization capability of our DQN+GNN agent in a wide range of real-world network topologies extracted from the Topology Zoo dataset [14]. From this dataset, we select the topologies that have more than 5 nodes and a maximum of 50 nodes. We intentionally excluded ring and star topologies from the dataset. The reason behind this is that the amount of valid candidate paths to allocate demands is typically very limited in these topologies (1 or 2 routing options in many cases). Thus, there is not significant room for optimization in such scenarios. To do this, first we filtered the topologies with an *average node degree* between 2 and 4. As a reference, NSFNet has an *average node degree equal to 3*. Then, to exclude ring topologies we computed the ratio between the *average node degree* and the *variance of the node degree* and picked those topologies with a ratio higher than 0.3. These values where selected based on a preliminary experimental analysis we made over the topologies in the dataset. As a result, our evaluation dataset contains 136 different topologies from the original dataset.

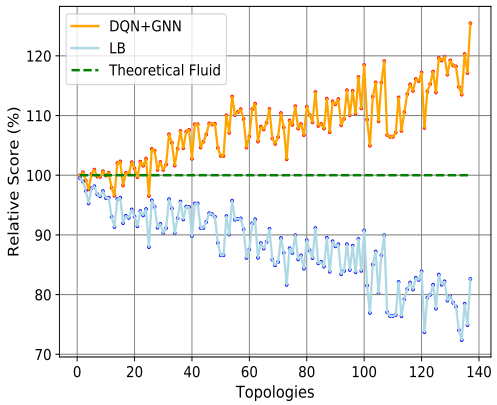


Figure 8: Evaluation of the proposed DQN+GNN agent in a dataset with real-world topologies.

To evaluate the generalization capabilities of our agent, we select the model with highest evaluation score during the training process in the NSFNet topology (Section 5.3). On each topology, we execute 1,000 evaluation episodes and store the average reward achieved by the DQN+GNN, the LB and the theoretical fluid routing strategies. Figure 8 shows the performance of our GNN-based DRL agent on the different topologies (x-axis). The topologies are ordered according to the difference of score between the DQN+GNN agent and LB policy. A list mapping the topology identifiers with the topology name in the Topology Zoo dataset will be publicly available upon acceptance of the paper. We compute the relative performance (in %) of our agent and the LB policy with respect to the theoretical fluid model. This figure reveals the ability of our DRL agent to maintain a good performance even when it operates in a routing scenario with real-world topologies not seen during training.

5.6 Use case

In this sub-section we present an use case to illustrate the potential of our DQN+GNN agent (Section 4) to perform relevant network optimization tasks. The use case consists in a scenario with the Geant2 topology where there are link failures. These failures change the original topology, and thus, modify the connectivity among nodes in the graph. We aim at evaluation if our agent is able to generalize also in the presence of link failures. When a link fails, it is necessary to find a new routing configuration that avoids this link. As the number of links failures increases, less paths are available and therefore the network may typically support less traffic volume.

We evaluate our DQN+GNN agent in routing scenarios where we remove up to 10 links from the Geant2 topology. Particularly, we make experiments on scenarios where the number of links that fail ranges from 0 to 10. In every case,

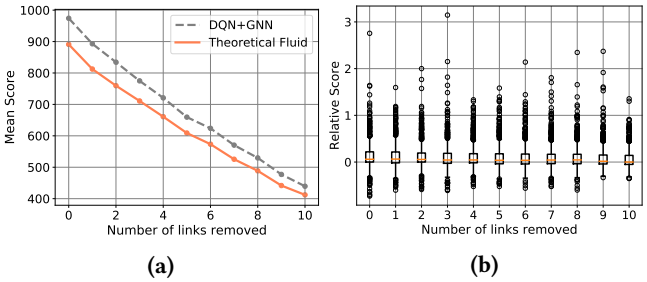


Figure 9: Evaluation in an use case with link failures

we execute 1,000 experiments where n links are randomly removed from the topology.

We compare the score obtained with respect to the theoretical fluid model. In Figure 9a we can observe the average score achieved over 1,000 experiments (y-axis) with respect to the number of link failures (x-axis). As more links fail, the score decreases because the total capacity of the network also decreases. The score difference between our agent and the theoretical fluid model decreases slowly as the number of failures increases. This suggests that our agent is robust to changes in the topology. In Figure 9b we observe the relative score of our DQN+GNN agent with respect to the theoretical model. In line with the previous results, the relative score is maintained as we remove shows that the DQN+GNN agent is able to generalize even in the worst case with 10 link failures.

6 RELATED WORK

Numerous DRL-based solutions have been proposed in the past to solve network optimization problems [9, 32]. In the networking field, finding the optimal routing configuration from a given traffic matrix is a fundamental problem, which is NP-hard. In this context, several DRL-based solutions have been proposed to address routing optimization [6, 8, 28]. However, they fail to generalize to unseen scenarios. They typically pre-process the data from the network state and presents it in the form of fixed-size matrices. Then, these representations are processed by traditional neural networks (e.g., fully-connected, convolutional neural networks). These neural networks are not suited to learn and generalize over data that is inherently structured as graphs. As a result, state-of-the-art DRL agents perform poorly when they are evaluated in different topologies that were not seen during the training.

7 CONCLUSION

In this paper, we presented a DRL architecture based on GNNs that is able to generalize to unseen network topologies. The use of GNNs to model the network environment allow the DRL agent to operate in different networks than

those used for training. We believe that the lack of generalization was the main obstacle preventing the deployment of existing DRL-based solutions in production networks. Thus, the proposed architecture is the first step towards the development of a new generation of DRL-based products for networking.

In order to show the generalization capabilities of our DRL+GNN architecture, we selected a classical problem in the field of OTN. This served as a baseline benchmark to validate the generalization performance of our model. Our results show that our model is able to sustain a similar accuracy in a network never seen during training. Previous DRL solutions based on traditional neural network architectures are not able to generalize to other topologies.

Our ongoing work is focused on applying our DRL+GNN architecture to more complex routing and networking problems. Given the generalization performance shown by GNNs for modeling more complex scenarios, we are confident similar results will also be obtained when combined with DRL.

ACKNOWLEDGMENTS

This work was supported by the Spanish MINECO under contract TEC2017-90034-C2-1-R (ALLIANCE), the Catalan Institution for Research and Advanced Studies (ICREA), by FI-AGAUR grant by the Catalan Government and by the Polish Ministry of Science and Higher Education with the subvention funds of the Faculty of Computer Science, Electronics and Telecommunications of AGH University. The research was also supported in part by PL-Grid Infrastructure.

REFERENCES

[1] 2019. ITU-T Recommendation G.709/Y.1331: Interface for the optical transport network. https://www.itu.int/rec/T-REC-G.709/.

[2] Fernando Barreto, Emílio CG Wille, and Luiz Nacamura Jr. 2012. Fast emergency paths schema to overcome transient link failures in ospf routing. *arXiv preprint arXiv:1204.2465* (2012).

[3] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. 2016. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*. 4502–4510.

[4] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261* (2018).

[5] Léon Bottou. 2010. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*. Springer, 177–186.

[6] Justin A Boyan and Michael L Littman. 1994. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Advances in neural information processing systems*. 671–678.

[7] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. arXiv:arXiv:1606.01540

[8] Xiaoliang Chen, Jiannan Guo, Zuqing Zhu, Roberto Proietti, Alberto Castro, and SJB Yoo. 2018. Deep-RMSA: A deep-reinforcement-learning routing, modulation and spectrum assignment agent for elastic optical networks. In *2018 Optical Fiber Communications Conference and Exposition (OFC)*. IEEE, 1–3.

[9] Gagan Choudhury, David Lynch, Gaurav Thakur, and Simon Tse. 2018. Two use cases of machine learning for SDN-enabled IP/optical networks: Traffic matrix prediction and optical path performance prediction. *IEEE/OSA Journal of Optical Communications and Networking* 10, 10 (2018), D52–D62.

[10] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*. 3844–3852.

[11] Miriam Di Ianni. 1998. Efficient delay routing. *Theoretical Computer Science* 196, 1-2 (1998), 131–151.

[12] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 1263–1272.

[13] Xiaojun Hei, Jun Zhang, Brahim Bensaou, and Chi-Chung Cheung. 2004. Wavelength converter placement in least-load-routing-based optical networks using genetic algorithms. *Journal of Optical Networking* 3, 5 (2004), 363–378.

[14] Simon Knight, Hung X Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan. 2011. The internet topology zoo. *IEEE Journal on Selected Areas in Communications* 29, 9 (2011), 1765–1775.

[15] Josue Kuri, Nicolas Puech, and Maurice Gagnaire. 2003. Diverse routing of scheduled lightpath demands in an optical transport network. In *Fourth International Workshop on Design of Reliable Communication Networks, 2003.(DRCN 2003). Proceedings*. IEEE, 69–76.

[16] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2015. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493* (2015).

[17] Shih-Chun Lin, Ian F Akyildiz, Pu Wang, and Min Luo. 2016. QoS-aware adaptive routing in multi-layer hierarchical software defined networks: A reinforcement learning approach. In *2016 IEEE International Conference on Services Computing (SCC)*. IEEE, 25–33.

[18] Albert Mestres, Eduard Alarcón, Yusheng Ji, and Albert Cabellos-Aparicio. 2018. Understanding the modeling of computer network delays using neural networks. In *Proceedings of the 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*. ACM, 46–52.

[19] Albert Mestres, Alberto Rodriguez-Natal, Josep Carner, Pere Barlet-Ros, Eduard Alarcón, Marc Solé, Victor Muntés-Mulero, David Meyer, Sharon Barkai, Mike J Hibbett, et al. 2017. Knowledge-defined networking. *ACM SIGCOMM Computer Communication Review* 47, 3 (2017), 2–10.

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[21] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[22] Krzysztof Rusek, José Suárez-Varela, Albert Mestres, Pere Barlet-Ros, and Albert Cabellos-Aparicio. 2019. Unveiling the potential of Graph Neural Networks for network modeling and optimization in SDN. In *Proceedings of the 2019 ACM Symposium on SDN Research*. ACM, 140–151.

[23] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE Transactions on Neural Networks* 20, 1 (2008), 61–80.

[24] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815* (2017).

[25] John Strand, Angela L Chiu, and Robert Tkach. 2001. Issues for routing in the optical layer. *IEEE Communications Magazine* 39, 2 (2001), 81–87.

[26] José Suárez-Varela, Sergi Carol-Bosch, Krzysztof Rusek, Paul Almasan, Marta Arias, Pere Barlet-Ros, and Albert Cabellos-Aparicio. 2019. Challenging the generalization capabilities of Graph Neural Networks for network modeling. In *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*. ACM, 114–115.

[27] José Suárez-Varela, Albert Mestres, Junlin Yu, Li Kuang, Haoyu Feng, Pere Barlet-Ros, and Albert Cabellos-Aparicio. 2019. Routing based on deep reinforcement learning in optical transport networks. In *Optical Fiber Communication Conference*. Optical Society of America, M2A–6.

[28] José Suárez-Varela, Albert Mestres, Junlin Yu, Li Kuang, Haoyu Feng, Albert Cabellos-Aparicio, and Pere Barlet-Ros. 2019. Routing in optical transport networks with deep reinforcement learning. *Journal of Optical Communications and Networking* 11, 11 (2019), 547–558.

[29] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. 2013. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*. 1139–1147.

[30] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[31] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3-4 (1992), 279–292.

[32] Zhiyuan Xu, Jian Tang, Jingsong Meng, Weiyi Zhang, Yanzhi Wang, Chi Harold Liu, and Dejun Yang. 2018. Experience-driven networking: A deep reinforcement learning based approach. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 1871–1879.