Routing with Graph Convolutional Networks and Multi-Agent Deep Reinforcement Learning

Sai Shreyas Bhavanasi
Computer Science
Saint Louis University

Lorenzo Pappone
Computer Science
Saint Louis University

Flavio Esposito
Computer Science
Saint Louis University

2022 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN) | 978-1-6654-7334-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/NFV-SDN56302.2022.9974607

***Abstract*—The computer networking community has been steadily increasing investigations into machine learning to help solve tasks such as routing, traffic prediction, and resource management. In particular, due to the recent successes in other applications, Reinforcement Learning (RL) has seen steady growth in network management and, more recently, in routing. However, changes in the network topology prevent RL-based routing approaches from being employed in real environments due to the need for retraining. In this paper, we approach routing as an RL problem with two novel twists: minimizing flow set collisions and dealing with routing in dynamic network conditions without retraining. We compare this approach to other routing protocols, including multi-agent learning, to various Quality-of-Service metrics, and we report our lesson learned.**

***Index Terms*—Routing protocols, Machine learning algorithms, Reinforcement learning, IP networks, Network Management.**

I. Introduction

An emerging application of Machine Learning (ML) is the automation of computer network management tasks. ML has been used recently for various tasks in networking ranging from traffic classification and resource management to attack detection, to name a few. However, several solutions in the networking field have recently proven that one of the most prolific ML areas is Reinforcement Learning (RL). In particular, traditional approaches to packet routing using RL leverage a single-agent learning approach to learn the environment. Despite such recent achievements in routing with reinforcement learning, a single agent still has to learn the topology through a centralized strategy, representing an impractical scenario for realistic environments. To cater to such needs, centralized techniques, such as graph convolutional networks or distributed RL approaches, have been recently designed, bringing several benefits. For example, in multi-agent RL, agents cooperate to achieve a shared objective, learn faster, and occasionally safeguard privacy. They may withstand failures overcoming the physical limitation of a single RL agent. For example, in multi-agent RL, agents cooperate to achieve a shared objective, learn faster, occasionally safeguard privacy, to a certain extent and may withstand failures overcoming the physical limitation of a single RL agent.

Prior traffic engineering solutions have focused on the improvement of routing algorithms using RL with interesting techniques, but without considering resiliency against network changes during or after the learning process; Other solutions, e.g., [1], ignored the need to explicitly improve path congestion awareness, and merely focused on learning to route.

In this paper, we approach the problem of routing with reinforcement learning with adaptability in mind. By adaptability, we mean an approach that applies to both intra and inter-domain routing problems, e.g., iBGP and eBGP, as well as adaptability to network changes without the need to retrain the RL algorithm. In some cases [2], to fine-tune traffic engineering policies, it is desirable to overwrite classical interior BGP routing rules, such as Equal Cost Multi-Path (ECMP) and Open Shortest Path First (OSPF). In other cases, a distributed approach is the only viable solution [3]. We present the design, implementation, and evaluation of two routing schemes based on RL, one using single-agent and one using multi-agent learning. Each one is suitable for a partially disjoint subset of computer networks. The Single-Agent RL uses Graph Convolutional Networks [4] to minimize retraining needs. The Multi-Agent RL instead uses Deep Q-Networks, where federated routing agents cooperate, suiting e.g., exterior BGP (eBGP) or Wide Area Networks under a single administrative domain.

Our *Single-Agent Graph Convolutional Network* algorithm (SA-GCN) operates directly on network traffic datasets encoded in a graph format, instead of a traditional vector and matrix data representations. The advantage of having a graph as input permits an RL policy to operate in any topology and learn from changes in routing and congestion events without the need to retrain.

In our *Multi-Agent routing with Deep Q-Network* algorithm instead (MA-DQN), each agent makes its routing decisions locally. By feeding the network topology to the RL agent's state space, the machine learning model is able to handle changes in latency and bandwidth more efficiently. Moreover, as RL agents learn about the network topology explicitly, the training time significantly improves, confirming our intuitions.

Among our main findings, we report that our RL models outperform the standard routing algorithms (with and without RL) and achieve an overall improvement under several metrics. Furthermore, we found that our MA-DQN routing algorithm achieves the optimal policy much faster than our single-agent counterpart (SA-GCN).

Our main contributions can hence be summarized as follows: we dissect the routing with reinforcement learning problem through (i) separately analyzing advantages and drawbacks of using both single-agent and multi-agent RL, (ii) giving an overall comparison between them in terms of network change resiliency and retraining needs. (iii) We

discuss each model to leave insights on the optimization process when tackling a challenging task such as network routing with changes and (iv) we compare their performance against existing routing algorithms.

The remainder of the paper is structured as follows. In Section II we overview the state-of-art approaches on RL for routing optimization, detailing both single and multi-agent solutions. Section III defines the RL model for GCN and DQN-based algorithms. Section IV reports the evaluation of the results in terms of the achieved QoS metrics and compared our approaches with the routing baselines. In Section V we conclude the paper.

II. Related Work

Routing protocols pose challenging requirements for ML models, such as dealing with and scaling complex and dynamic network topologies, learning the correlation between the selected path and the perceived QoS, and the ability to forecast the repercussions of routing decisions. Different techniques to apply RL to the traffic routing problem have been proposed in literature [5, 6, 7, 8]. These techniques differ in terms of (i) learning capability distribution and (ii) the amount of collaboration among numerous learners. The presence of a central node —the controller in Software-Defined Networks (SDN) and the sink in Internet of Things (IoT) networks, respectively — allows for centralized learning in SDN [9] and IoT. Routing in IoT, on the other hand, necessitates decentralized RL [10, 11], with the learning capability dispersed across the routing nodes. In the rest of this section, we focus on solutions that most closely match our contribution, with respect to two dimensions: Single-Agent Q-learning solutions for traffic engineering and routing with Multi-Agent learning.

A. Single-Agent Deep Q-learning for Traffic Engineering

A few recent solutions have employed Deep Q-Learning to tackle traffic engineering problems. The authors in [12] implement a deep-RL solution to improve the performance of baseline TE algorithms of an SD-WAN-based network in terms of service availability. Results show that Deep Q-Learning achieves better performance with respect to the other deep-RL algorithms and the baselines in terms of the percentage of time in which the service is up. In [13], a greedy online QoS routing method based on dueling deep Q-network is employed, proving that this solution can learn the network topology to solve multiple QoS metrics optimization tasks.

Differently from all these sound solutions, in this work, we investigate the application of GCNs to Deep Q-Learning for a QoS routing optimization problem.

B. Routing with Multi-Agent Reinforcement Learning

Although previous studies of DRL-based techniques have demonstrated the ability to deploy routing configurations in dynamic networks autonomously, some researchers have argued that centralized controller approaches are likely to face challenges in large-scale networks due to the difficulties of collecting widely distributed network status in real time.

Multi-Agent Reinforcement Learning has rapidly become an important research direction. A few authors have proposed bringing these approaches to optimize routing protocols [14, 15, 16, 17]. In particular, [16] proposes a multi-agent reinforcement learning framework for adaptive routing in communication networks, which takes advantage of both the real-time Q-learning and the actor-critic methods.

Pinyoanuntapong *et al.* [15] formulated the traffic engineering decision-making problem as a Multi-Agent Markov decision process (MA-MDP) In MA-MDP, observations, actions, and rewards of multiple agents are integrated for joint learning. Similar to these approaches, we use the Q-routing technique for traffic-aware routing, but our solution uses a fully distributed multi-agent Deep Q-Learning Reinforcement Learning algorithm to deal with QoS requirements and topological changes.

III. Model and Background on GCN and MA-DQN

In this section, we detail the design of our single-agent and multi-agent routing approaches. Specifically, we describe the RL model and define the state and action space along with the selected reward functions both for GCN-based algorithm (Section III-A) and DQN (Section III-B).

A. Single Agent GCN Policy Background and Settings

GCNs (Graph Convolutional Networks) [4] are a graph-based variation of Convolutional Neural Networks (CNNs). GCNs stack layers of learned first-order spectrum filters followed by a nonlinear activation function to learn a graph representation.

The key benefit of using GCNs as the policy network is the policy's ability to not be restricted to one network topology. The topology is explicitly provided as an input to the policy network. This topology agnostic approach is demonstrated in our experiments. A network experiences congestion when several flow sets coexist on the same underlying path. *We aim at learning how to minimize the coexistence of flow sets as long as alternative routes exist. We define such flow coexistence to be a collision, when two or more flow sets coexist in the same forwarding application process, i.e, a (virtual) router or switch, at the same time.*

Since an action that an RL agent chooses influences subsequent actions, we model the problem as a Sequential Decision-making Problem (SDP). The following tuple characterizes an instance of our problem: $M =< S, A, R, \gamma >$, where S is a finite set of states, A is a finite set of actions, R is the immediate reward, and γ is the discount factor. We denote f as the number of flow sets in the system, and N is the number of nodes in the system. We design our RL algorithm by leveraging an "on-policy approach", aiming to optimize our policy network output (i.e., the probability distribution of actions). In order to achieve a more stable training and avoid divergence, we use the PPO (*Proximity Policy Optimization* [18]) algorithm. We next describe each element of the Sequential Decision-making Problem tuple in detail.

State Space. Let S denote the finite set of states that are admissible in the environment. An arbitrary state contains three parts. The first part is the adjacency matrix of the network. The second part is a 2-column matrix containing a one-hot encoding of the source and destination nodes of the flow set to be routed by the GCN. The third part is a $f \times 2$ matrix where each layer of the matrix is a similar 2-column matrix but of a competing flow set in the network. The state does not contain what path other flow sets are taking across the network but merely their source and sink nodes.

Action Space. Let A denote the finite set of actions that the agent can take. As in [19], we construct the action space as a one-hot vector the size of the highest degree node in the graph.

Reward Functions. Let $R(s, a)$ denote the immediate reward (or expected immediate reward) received for selecting an action (routing decision) $a \in A$ at a state $s \in S$ which causes the state transition $s \to s'$, and $R(s, a) \in [-1, 1]$.

$$R(s,a) = \begin{cases} -1 & \text{could not reach destination} \\ \text{r} & \text{reached destination} \\ 0.1 & \text{moved closer to target} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The environment in which the agent operates receives a one-hot encoding of the destination as input. The input is then passed onto a feed-forward neural network with a ReLU activation. The input layer has a size equal to the number of nodes in the graph. Subsequently, there are two hidden layers of size $\frac{|V|}{2}$ followed by an output layer with a size equal to the number of neighbors for that agent.

B. Multi Agent DQN Background and Settings

In our MA-DQN approach, we design an algorithm that uses a Deep Q-Network (DQN), a combination of reinforcement learning techniques and deep neural networks [20].

To stabilize the training, we first employ a replay buffer in our algorithm that keeps track of the (s, a, r, s', a') tuple, where s denotes the current state, a denotes the current action, r denotes the current reward, s' denotes the next state, and a' denotes the next action. Secondly, DQN uses an iterative update that periodically adjusts action values to the target values to avoid correlations.

The computer network in which the RL agent operates is described by a standard graph $G(V, E)$ where V represents the set of routing nodes and E represents the set of transmission links. Each transmission link contains some traffic that we simulate. The goal is to find the path that minimizes the latency between each source (s) and destination (d).

In the MA-DQN model, each node is an agent that implements DQN independently. Each agent has its neural network and makes routing decisions independently.

State Space. To train our RL model, we construct the state space as a one-hot encoding vector whose size equals $|V|$, i.e., the number of routers. The one-hot vector represents the destination of the packet. In each episode, the state space remains similar among all agents.

Action Space. We construct the action space also as a one-hot vector whose size equals the number of neighbors of each agent. At each time step t, the agent chooses a neighbor after observing the state space.

Reward Functions. To route in larger networks, we use a dense reward function. Per our reward function in Equation 2-3, the agent receives either a positive or a negative reward for each routing decision taken. We find that this structure significantly increases the rate at which the model converges.

Therefore, if the packet is more than one step away from the destination:

$$R(s,a) = \begin{cases} a & \text{moved closer to destination} \\ -.75 & \text{otherwise} \end{cases} \tag{2}$$

Where a is the difference between the time it would take the packet to reach the destination from the node it is currently at, and the time it would take for the packet to reach the destination from the node it was at. If the packet can reach the destination using one link, the reward structure is:

$$R(s,a) = \begin{cases} 1.5 & \text{took best link to destination} \\ -.75 & \text{took sub-optimal link to destination} \end{cases} \tag{3}$$

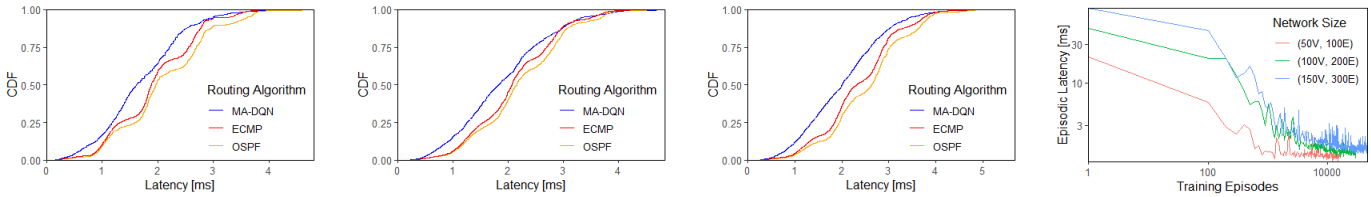
The environment in which the agent operates receives a one-hot encoding of the destination as input. The input is then passed onto a feed-forward neural network with a ReLU activation. The input layer has a size equal to the number of nodes in the graph. Subsequently, there are two hidden layers of size $\frac{|V|}{2}$ followed by an output layer with a size equal to the number of neighbors for that agent.

IV. Evaluation

In this section, we evaluate both proposed algorithms, single-agent via GCN and multi-agent via DQN, with respect to Quality of Service (QoS) metrics for traffic engineering.

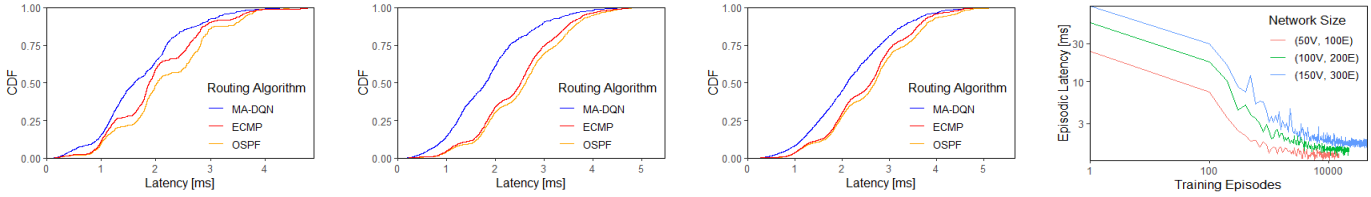
A. Evaluation Settings

In all our experiments, we considered network topologies following both Waxman (random-biased) and Barabasi-Albert (preferential attachment) topologies. Our networks have sizes 50 Vertices, 100 edges (50V, 100E), (100V, 200E), and (150V, 300E). These topologies were generated using the BRITE topology generator [21]. After generating these network topologies, we simulate several traffic scenarios in static and dynamic conditions. In particular, in our experiments, we simulate traffic by generating flows that take the shortest path from randomly generated source-destination pairs. For each packet traveling on a (virtual) link, we reduce the corresponding residual capacity and increase the latency proportionally to the flow packet size. Unless otherwise specified, we use a replay buffer size of 5000, a batch size of 192, and the RMSprop optimizer. Specifically, in our SA-GCN solution, we improve vanilla Q-Routing by implementing two different approaches: Double Q Learning [22] and Dueling DQN [23].



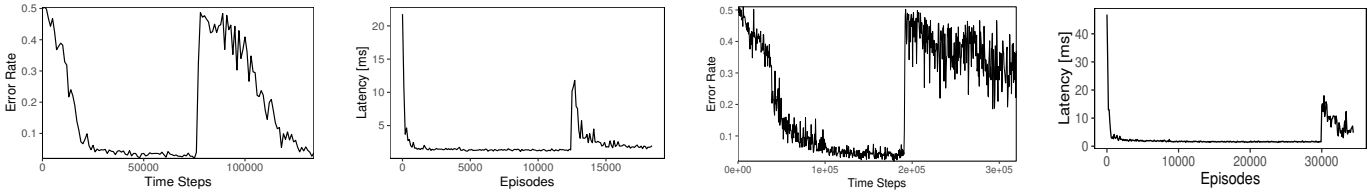
(a) (b) (c) (d)

Figure 1: CDF of the latency of a converged MA-DQN model on (a) (50V, 100E), (b) (100V, 200E), and (c) (150V, 300E) Barabasi-Albert Network. We observe that the MA-DQN model can consistently outperform OSPF and ECMP. (d) Episodic latency of the MA-DQN model during training in Barabasi-Albert Networks.



(a) (b) (c) (d)

Figure 2: CDF of the latency of a converged MA-DQN model on (a) (50V, 100E), (b) (100V, 200E), and (c) (150V, 300E) Waxman Network. (d) Episodic latency of the MA-DQN model during training in Waxman.



(a) (b) (c) (d)

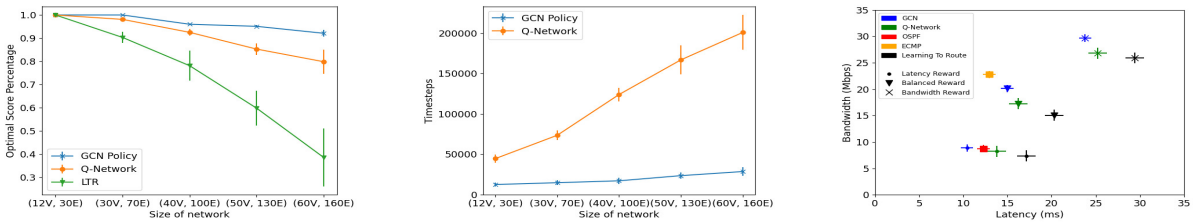
Figure 3: (a) Evaluating the MA-DQN model's accuracy during training when 5 nodes of a (50V, 100E) Barabasi topology is removed after the model converges (b) Evaluating the latency for the same experiment (c) Evaluating the model's accuracy when 5 nodes from a (100V,200E) Barabasi-Albert network is removed. (d) Evaluating the model's latency for the same experiment.

B. Evaluation of Multi Agent DQN

1) Multi Agent DQN based policies outperforms OSPF and ECMP: In this section, we compare the latency of converged DQN policies, OSPF and ECMP.

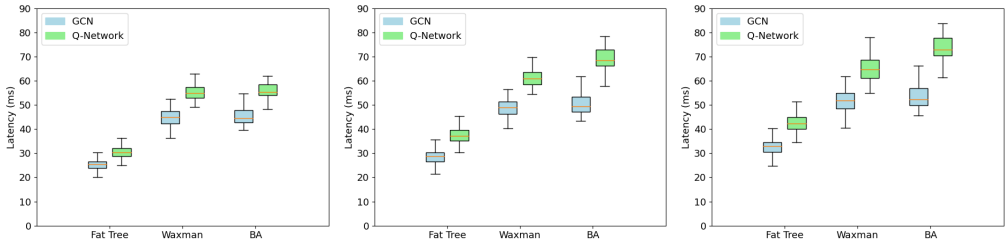
Figures 1(a-c) and 2(a-c) compare the latency of the aforementioned policies on networks of various sizes and topology, showing that Multi-Agent DQN based policies learn near-optimal routing policies with networks of sizes up to (150V, 300E). One of the most significant results from our experiments is the model's scalability, as we found that our solution outperforms classically adopted routing protocols such as OSPF and ECMP even on large networks. From Figures 1(d) and 2(d) we observe that the training time scales well with respect to the topology size. We attribute this to the fact that the network topology is built into the neural network of each agent. Our application of a dense reward function plays a significant role in performance.

2) Impact of Retraining Needs: In this experiment set, we further investigate the performance of the model when the topology changes. We observe the model's performance when a number of nodes are removed (become unavailable) after the model reaches convergence. The results in Figure 3(a-b) are generated from a 50 Node network following the Barabasi-Albert model. For each experiment trial, we remove 5 nodes at random after the RL model training had converged. We route 2000 packets after the model is fully trained. As expected, after removing the nodes, the model can relearn the optimal routing strategy. However, for larger networks the re-learning time grows significantly. Figures 3(c-d) show that when removing 5 nodes from a (100V, 200E) Barabasi-Albert network topology after the model converges, the model is unable to relearn an optimal policy quickly, *suggesting that it is faster to retrain the model from scratch on larger networks especially since the model converges relatively quickly when trained from scratch.* Since the neural network retains the environment's topology,



(*a*) (*b*) (*c*)

Figure 4: (a) Comparing the reward strength between a GCN policy, a Q-Network, and the reward used in the Learning to Route paper. (b) Comparison between number of training timesteps required for convergence. (c) Analysis of reward functions with respect to latency and bandwidth in a (150V, 350E) Waxman topology.



(*a*) (*b*) (*c*)

Figure 5: Analysis of latency for a GCN policy and Q-Network for three different network topology classes: Fat Tree, Waxman, and Barabasi-Albert (BA). (a): (50V, 100E), (b): (100V, 200E), and (c): (150V, 350E).

reduction of nodes resulted in a topology change too large for the machine learning model to relearn an optimal policy.

C. Evaluation of GCN

1) GCN based Policies leads to Faster Convergence and Stronger Reward Signals: In this experiment set, we compare the characteristics of converged GCN-based policies, Q-routing policies, and the policy used by the Learning to Route paper [1], used as benchmark (Figure 4).

We used a test set of a hundred different topologies for every tested algorithm for every network size. Once the model is fully trained, we test the model using 200 source-destination pairs for the SA-GCN model. The source and destination are chosen randomly.

Figures 4-a shows the optimal score percentage between our solution and benchmarks, varying the network size. We can see how our SA-GCN approach outperforms the others when optimizing the reward to prioritize low-latency paths. Figure 4-b compares the time steps required for policy convergence. Such convergence requires the policy to successfully route every source-destination in the test set of 100 randomized topologies for every network size.

Lastly, in Figure 4-c, we explore how various reward functions operate with respect to throughput and latency on a Barabasi-Albert graph sized (100V, 350E). For additional comparison, the performance of two of the most commonly used iBGP routing protocols, OSPF and ECMP, are plotted as well. The three reward structures explored are as follows: (i) optimize latency only, (ii) optimize bandwidth only, and (iii) co-optimize latency and bandwidth. As shown, each algorithm was capable of optimizing what each respective reward function sought to be optimized, showing that RL-based approaches can potentially be customized according to a particular set of QoS metrics.

2) Latency: In Figure 5 we show the network performance in terms of latency on a test set of 100 networks for each topology type. We selected a Fat Tree topology to simulate data center networks. Waxman and Barabasi-Albert topology generators were used for the other two topology classes. It is worth noticing that for each network topology and all the three sizes, a trained GCN-based policy outperforms a trained Q-Network in both median latency and standard deviation.

V. Conclusion

In this paper, we explored packet routing with reinforcement learning with a few novel twists. Our objective has been to study the impact of topology and traffic changes on a trained neural network. To do so, we focused on single domain routing, suitable e.g., for interior BGP, and on multi-domain routing, valuable for larger scale routing protocols such as exterior BGP. We proposed a Single Agent RL model, based on a Graph Convolutional Network (GCN), to fit the former, and a Multi-Agent Deep Q-Learning Network model for the latter. We evaluate both single and multi-agent solutions with different network size and connectivity models. We found that single-agent with GCN improves on the ability to achieve high

QoS-metrics when the computer network topology changes after the RL training had converged. We also found that our multi-agent model is able to scale well with larger networks, consistently outperforming OSPF and ECMP.

ACKNOWLEDGEMENTS

This work has been supported by NSF Awards # 1836906 and # 1908574. We would like to thank our former SLU undergraduate student Hunter Parks for his initial work on this project. Sai Shreyas Bhavanasi is an undergraduate student.

REFERENCES

[1] Asaf Valadarsky et al. "Learning to Route with Deep RL". In: *31st Conference of Neural Information Processing Systems*. 2017.

[2] Brandon Schlinker et al. "Engineering Egress with Edge Fabric: Steering Oceans of Content to the World". In: SIGCOMM '17. Los Angeles, CA, USA: ACM, 2017, 418–431. ISBN: 9781450346535.

[3] T. G. Griffin, F. B. Shepherd, and G. Wilfong. "Policy disputes in path-vector protocols". In: *Proceedings. Seventh International Conference on Network Protocols*. 1999, pp. 21–30. DOI: 10.1109/ICNP.1999.801912.

[4] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *Proceedings of the 5th International Conference on Learning Representations* (ICLR). Palais des Congrès Neptune, Toulon, France, 2017.

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

[6] Michael Littman and Justin Boyan. "A distributed reinforcement learning scheme for network routing". In: *Proceedings of the international workshop on applications of neural networks to telecommunications*. Psychology Press. 2013, pp. 55–61.

[7] Samuel PM Choi and Dit-Yan Yeung. "Predictive Q-routing: A memory-based reinforcement learning approach to adaptive traffic control". In: *Advances in Neural Information Processing Systems*. 1996, pp. 945–951.

[8] Zoubir Mammeri. "Reinforcement Learning Based Routing in Networks: Review and Classification of Approaches". In: *IEEE Access* 7 (2019), pp. 55916–55950. DOI: 10.1109/ACCESS.2019.2913776.

[9] Shih-Chun Lin et al. "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. 2016, pp. 25–33.

[10] Abhijeet A Bhorkar et al. "Adaptive opportunistic routing for wireless ad hoc networks". In: *IEEE/ACM Transactions On Networking* 20.1 (2011), pp. 243–256.

[11] Zhichu Lin and Mihaela van der Schaar. "Autonomic and distributed joint routing and power control for delay-sensitive applications in multi-hop wireless networks". In: *IEEE Transactions on Wireless Communications* 10.1 (2010), pp. 102–113.

[12] Sebastian Troia et al. "On deep reinforcement learning for traffic engineering in sd-wan". In: *IEEE Journal on Selected Areas in Communications* (2020).

[13] Syed Qaisar Jalil, Mubashir Husain Rehmani, and Stephan Chalup. "DQR: Deep Q-Routing in Software Defined Networks". In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2020, pp. 1–8.

[14] Xinyu You et al. "Toward Packet Routing with Fully-distributed Multi-agent Deep Reinforcement Learning". In: *CoRR* abs/1905.03494 (2019). arXiv: 1905.03494.

[15] Pinyarash Pinyoanuntapong, Minwoo Lee, and Pu Wang. "Delay-optimal traffic engineering through multi-agent reinforcement learning". In: *Proc. of IEEE INFOCOM Workshops*. IEEE. 2019, pp. 435–442.

[16] Siliang Zeng, Xingfei Xu, and Yi Chen. "Multi-Agent Reinforcement Learning for Adaptive Routing: A Hybrid Method using Eligibility Traces". In: *2020 IEEE 16th International Conference on Control Automation (ICCA)*. 2020, pp. 1332–1339.

[17] R. Rudek, L. Koszalka, and I. Pozniak-Koszalka. "Introduction to multi-agent modified Q-learning routing for computer networks". In: *Advanced Industrial Conference on Telecommunications/Service Assurance with Partial and Intermittent Resources Conference/E-Learning on Telecommunications Workshop (AICT/SAPIR/ELETE'05)*. 2005, pp. 408–413.

[18] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[19] T. Hu and Y. Fei. "QELAR: A Machine-Learning-Based Adaptive Routing Protocol for Energy-Efficient and Lifetime-Extended Underwater Sensor Networks". In: *IEEE Transactions on Mobile Computing* 9.6 (2010), pp. 796–809. ISSN: 2161-9875.

[20] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533.

[21] A. Medina et al. "BRITE: an approach to universal topology generation". In: *Proc, of MASCOTS*. 2001, pp. 346–353.

[22] Hado Hasselt. "Double Q-learning". In: *Advances in neural information processing systems* 23 (2010).

[23] Ziyu Wang et al. "Dueling network architectures for deep reinforcement learning". In: *International conference on machine learning*. PMLR. 2016, pp. 1995–2003.