Reinforcement Learning for Virtual Network Embedding in Wireless Sensor Networks

Haitham Afifi and Holger Karl
Computer Networks Group
Paderborn University, Germany
{haitham.afifi, hkarl}@ieee.org

***Abstract*—Upcoming sensing applications (acoustic or video) will have high processing requirements not satisfiable by a single node or need input from multiple sources (e.g., speaker localization). Offloading these applications to cloud or mobile edge is an option, but when running in a wireless senor network (WSN), it might entail needlessly high data rate and latency. An alternative is to spread processing inside the WSN, which is particularly attractive if the application comprises individual components. This scenario is typical for applications like acoustic signal processing.**

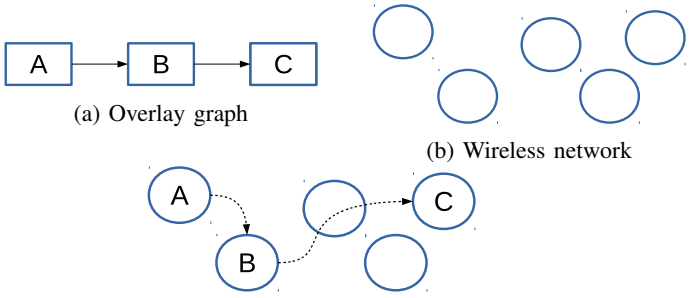
Mapping components to nodes can be formulated as wireless version of the NP-hard Virtual Network Embedding (VNE) problem, for which various heuristics exist. We propose a Reinforcement Learning (RL) framework, which relies on Q-Learning and uses either Greedy Epsilon or Epsilon Decay for exploration. We compare both exploration methods to the result of an optimization approach and show empirically that the RL framework achieves good results in terms of network delay within few number of steps.

I. Introduction

The advances in embedded devices have opened the door for a new class of applications in Internet of Things (IoT) and more specifically Wireless Sensor Networks (WSNs). First, the embedded devices are no longer limited to data collection, but they can do data processing. This can off-load the computation from central cloud services to end-user devices, reduce end-to-end latency, and add an extra privacy layer [1]. Second, the processing per node is not restricted to a single service or function. Thanks to virtualization, devices can run services on-demand, which facilitates the process of reconfiguration, fault management, hardware deployment, and decreases the cost.

Nevertheless, relatively heavy data-processing applications will over-utilize the processing of a single device. Instead, we can rely on a network of devices to process the data. An example of these applications is realized when distributing the computation in a WSN, i.e., a group of devices collaborate together for local computation, which is, indeed, a special case of fog computing [2]. Within this context, some work has considered locally distributing network functions [3], complex-data analysis [4], and acoustic signal processing [5].

Instead of having a monolithic format, an application is given by a set of processing functions, linked together in a predefined order. The main challenge appears when running the application inside a cluster of wireless nodes; we need to place each function on a node and find paths between the processing nodes (Figure 1). Selecting nodes for placement and paths for routing simultaneously is an NP-hard problem, which is known as Virtual Network Embedding (VNE) [6].



(a) Overlay graph (b) Wireless network

(c) VNE solution

Fig. 1: Visualization of VNE graphs

This approach has received a lot of interest because it exploits the idle devices for pipeline processing. Accordingly, it improves the overall processing performance compared to running on a single node as in edge computing [7]. There has been some work [8] that solves wireless VNE using optimization problem [9], [10], heuristics [11], [12] and meta-heuristics [13]–[15]. The objective for each solution differs between minimizing the packet losses, maximizing the throughput, minimizing the delay, or generally maximizing Quality of Services (QoS).

Different solutions with common objectives may assume different types of inputs, which makes it hard to compare between them. For instance, the work in [16] assumes that the data rate between wireless nodes is predefined, while the work in [11] assumes that the data rate is dependent on the interference from neighboring transmitting nodes. Hence, the comparison requires additional overhead to adapt either solution to the other.

Another family of solutions is Reinforcement Learning (RL), which requires less overheard for adapting one solution to another. It relies on standardized components and processes (Section II), therefore, subtle differences in the problem definition or solution attempts, can be reproduced by modifying or replacing the components and processes.

Nevertheless, it has not been well investigated for wireless

This work was supported by *Deutsche Forschungsgemeinschaft* (DFG) under contract no. KA2325/4-1 within the framework of the Research Unit FOR2457 “Acoustic Sensor Network”

VNE in particular. Most of the RL work focused on building paths between the nodes [17], without taking into consideration the placement aspect. The work in [18] considered reconfiguring existing placement solutions, under the assumption of changing resource requirements, but in our problem, we do not assume the existence of an initial placement solution, nor a change in the resource requirements. Reference [19] considered the placement aspect in mobile edge computing (MEC), in contrast to our case, they consider the placement of separate function rather than a graph of functions.

To sum up, existing RL solutions do not address the same facets of our problem and cannot be directly applied to our input assumptions for wireless networks. In this work, we formulate the placement problem using RL with corresponding states and actions. Then, we integrate the RL with a heuristic solution for a complete VNE solution. Finally we compare the RL framework to the optimization problem. Furthermore, we make our framework publicly available to make it easier for future comparisons and modifications [20].

II. Reinforcement Learning

Reinforcement Learning (RL) is an area of machine learning that works by taking *actions* to maximize the *reward* in a particular situation or *environment* [21]. Unlike the supervised learning, which needs in advance the knowledge about labels or the answer key, RL trains an *agent* to learn from trial and error without any prior knowledge about the ground truth. Hence, it makes it easier to adapt a solution to different objectives by just updating the reward.

For each action the agent takes, the environment reports back a reward and the new *state* for the agent. The rewards can be both good or bad to indicate positive or negative behaviors, respectively. Using a series of consecutive actions, the agent aims at maximizing the cumulative reward.

There are different classifications for reinforcement [22] learning algorithms, such as model-based vs. model-free and on-policy vs off-policy. Model-based algorithms assume that an action takes time to be executed, thus the agent uses this time to simulate a *model* to learn more about the environment given the previously taken actions and corresponding rewards. This becomes, however, impractical when the state and action space is large. Unlike model-based algorithms, model-free solution learn directly by trial and error and does not require the relatively large memory.

Each state has expected long-term reward. On the one hand, on-policy algorithms learn this reward by taking actions based on a current policy. On the other hand, off-policy algorithms learn the reward by taking actions that do not necessarily follow the current policy. In this paper, we use Q-learning as a model-free off-policy algorithm to solve the VNE problem .

Initially, the agent has no knowledge about the environment, hence, the selection of an action is based on two main processes: *Exploration* and *Exploitation*. The former is used to collect information about the environment, even if it means selecting an action with a bad reward. The latter uses the collected information and selects the best action with the maximum reward. The trade-off between exploration and exploitation is dependent on the problem definition and must be carefully balanced, accordingly.

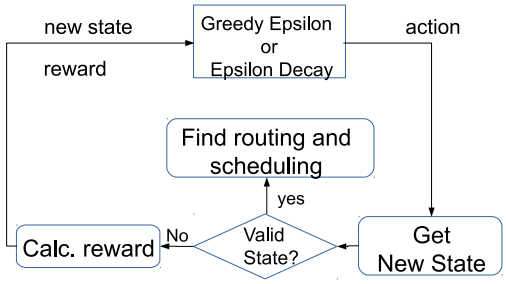


Fig. 2: RL framework describing the role of agent and the process of environment as described in Section IV-A

We follow an episodic formulation to implement and evaluate the RL algorithm. It defines *step* t, at which an agent performs an action and the environment returns the new state and reward. It also defines an *episode*; as a sequence of steps (i.e., actions, states and rewards) that ends with a terminal state.

We use Q-learning method to find the next action, which is selected based on a greedy behavior. Consequently, a Q-table containing Q-values (per each state S_t and action A_t) is created for the given environment. We initially assign same negative values to the Q-table, then the Q-value is updated using:

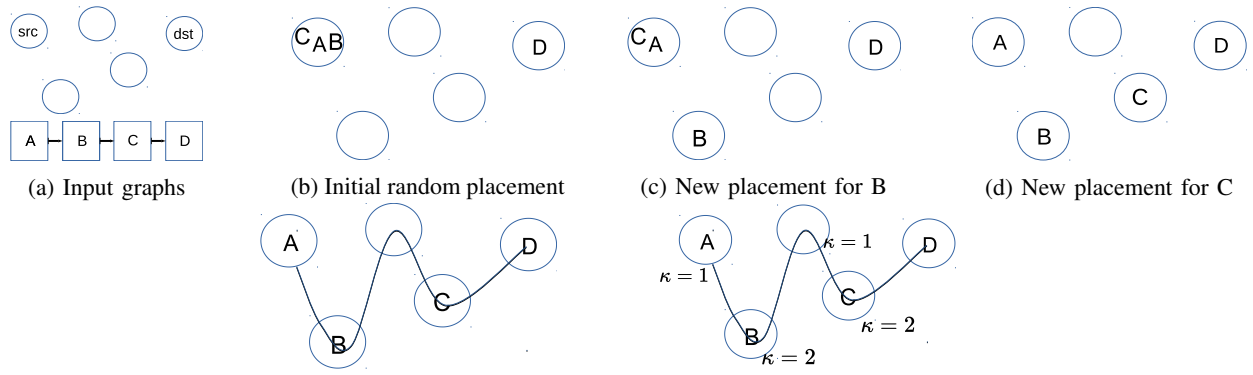
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[G_t + \gamma \arg\max Q(S_{t+1}, a) \right] \quad (1)$$
$$\text{and } G_t = R_t - Q(S_t, A_t),$$

where $\alpha \in [0, 1]$ is defined as the learning rate; the extent to which Q-values are updated per each step, and $\gamma \in [0, 1]$ is the discount factor that determines the importance given to future rewards; $\gamma = 0$ will make the agent short-sighted by only considering current reward R_t.

Indeed, the Q-values are a mapping of state-action pairs to the expected returns, which are updated by adding the old Q-values with the new learned ones. Although the Q-learning seems to be always greedy by selecting the maximum Q value for future states, this is not always the case. An additional meta-parameter, *epsilon* ($\epsilon \in [0, 1]$), is used to control the exploration rate; when $\epsilon = 1$ an action is always selected at random.

III. System Model

We give a brief overview of the problem at hand. First, we define the inputs as the *infrastructure* and *overlay* graphs (Figure 3a). The infrastructure graph specifies the wireless nodes $v \in V$, their processing capacities c_v and the attenuation between the nodes $a_{v,v'}$. Additionally, we assume a contention-free time-based MAC protocol and set a maximum set of timeslots K that can be used within a time frame and a common noise floor N_0. Furthermore, we notate v_{src} and



(a) Input graphs (b) Initial random placement (c) New placement for B (d) New placement for C

(e) Find shortest paths (f) Assign transmission timeslots

Fig. 3: Visualizing the RL framework steps; assume all nodes can run one process at maximum.

TABLE I: Summary of Variables.

Variable	Description
$\theta(p, v)$	$\in \{0, 1\}$ for placing a function p on node v
$f(v, \kappa)$	$\in \{0, 1\}$ to determine if node v is transmitting at timeslot κ
$s(v_1, v_2, p, \kappa)$	$\in \{0, 1\}$ states that v_1 sends to v_2 the output of processing function p at timeslot κ
$h(v_1, v_2, p)$	$\in \mathbb{N}$ to determine how many tasks needs to receive the p when node v_1 sends it to v_2

v_{sink} as the source (e.g., microphone sensors) and sink (e.g., gateway or database) nodes, respectively.

The overlay graph denotes the distributed application as a directed graph; it consists of processing virtual functions $p \in P$ and the required node resources per each processing function w_p. The functions are connected via virtual links $l = (p, p')\forall \in L$, where each link requires a data rate that is mapped to a threshold signal-to-interference ratio SINR_l. For simplicity, we assume in this paper that all links require the same data rate, therefore, they require the same SINR_{th}. Similar to the infrastructure graph, we also define p_{src} and p_{sink} to identify the start and end processing functions of the application.

In order to embed the overlay graph into the infrastructure graph, there are a couple of constraints that should not be violated (see the variables in Table I).

The capacity constraint (Eq. (2)) ensures that a node is not overutilized by the running processes. The duplex constraint (Eq. (3)) ensures that a node does not send and receive at the same timeslot. The flow control constraints (Eq.(4) – (6)) ensure that flow conservation rules for packet forwarding in a multicast environment is not violated. The interference constraint (Eq. 7) ensures that the SINR for a receiving node at a timeslot is above the SINR_{th}. For more details about the problem definition, we refer to [11].

$$\sum_{b \in B} \theta(b, v) \cdot w_b \leq c_v, \forall v \in V \tag{2}$$

$$f(v, \kappa) + \sum_{v_i \in V} \sum_{p \in P} s(v_i, v, p, \kappa) \leq 1, \begin{matrix} \forall v \in V \\ \forall \kappa \in K \end{matrix} \tag{3}$$

$$\sum_{v' \in V} h(v_{\text{src}}, v', b_{\text{src}_p}) - \sum_{(b_{\text{src}_p}, b') \in L} \theta(b_{\text{src}}, v_{\text{src}}) = 0\,, \qquad \forall b_{\text{src}_p} \in b_{\text{src}_P} \tag{4}$$

$$\sum_{v' \in V} \sum_{(p, b_{\text{sink}}) \in L} h(v', v_{\text{sink}}, p) - \sum_{(b_p, b_{\text{sink}}) \in L} \theta(b_{\text{sink}}, v_{\text{sink}}) \geq 0 \tag{5}$$

$$\sum_{v' \in V} h(v, v', p) - \sum_{(p, b')} \theta(b, v) = \sum_{v' \in V \setminus v} h(v', v, p) - \sum_{(p, b')} \theta(b', v), \forall p \in P \setminus \{b_{p_{\text{sink}}}, b_{p_{\text{sink}}}\}^{\forall v \in V} \tag{6}$$

$$\sum_{p \in P} s(v, v', p, \kappa) \cdot \text{SINR}_{\text{th}} \leq \frac{f(v, \kappa)}{N_o + I(v, v')} a v, v', \begin{matrix} \forall \{v, v'\} \in V \\ \forall \kappa \in K \end{matrix}$$

$$\text{where } I(v, v') = \sum_{\substack{u \in V \\ u \neq v}} f(u, \kappa) \cdot \gamma_{u, v'} \tag{7}$$

IV. Problem Formulation

In this section, we describe the formulation of states and actions, as well as the implementation of the environment and agent of the RL framework. (Figure 2).

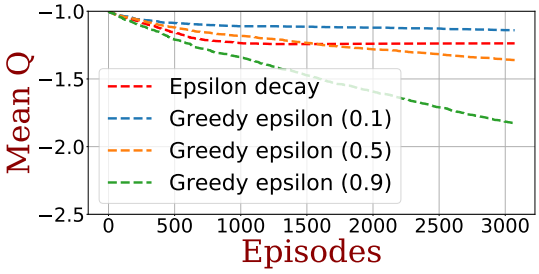
A. States and Actions

We represent the state space of the embedding environment by the placement of virtual functions on the infrastructure nodes. Consequently, a state is given by an array of length $|B|$, where each element of this array has a value $\in V$; the total number of states is given by $|V|^{|B|}$.

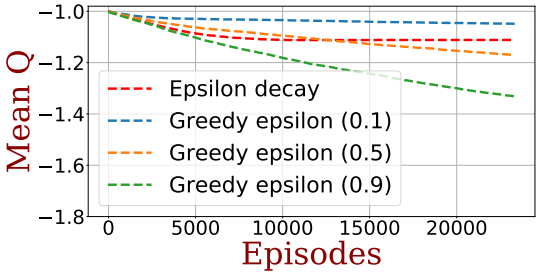
The placement solution per state is not necessarily always valid (e.g., the capacity constraint may be violated as in Figures 3b and 3c). Therefore, the agent takes an action to move the placement of a virtual function from one node to another (Figures 3c and 3d); the total number of actions per state is equal to $|V| \cdot |B|$.

B. Reward Function

The reward for every state-action is based on two main criteria: First, whether the new state's solution satisfies all constraints (Figure 3d). Second, the number of timeslots required to send the data from source to destination, based on the placement solution (Figure 3f).

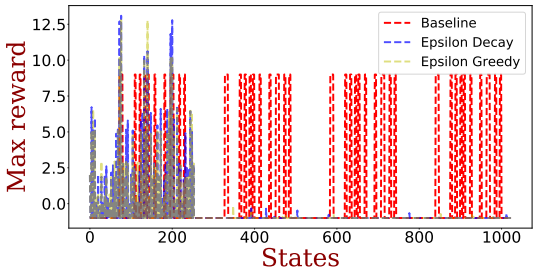


(a) 4 Nodes

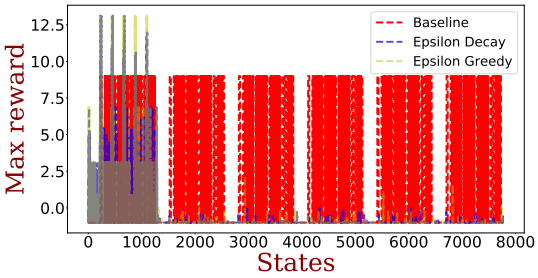


(b) 6 Nodes

Fig. 4: Mean Q-value convergence



(a) 4 Nodes



(b) 6 Nodes

Fig. 5: Explored maximum reward per state

Each state yields a reward according to its placement solution. If the solution violates any of the constraints (Section III), a negative value ($-r_\zeta$) is given every time a constraint is violated (Eq. (8))

$$R_t = -r_\zeta \tag{8}$$

If the state has a valid solution, the shortest path between the nodes used for placement is calculated, where a path is calculated for each link in the overlay graph (Figure 3e). Then, we estimate the number of timeslots needed to send the data along the paths, by solving for the duplex (Eq. (3)) and interference (Eq. (7)) constraints. Finally, we terminate the episode and calculate the reward using Eq. (9)

$$R_t = \frac{|B| \cdot r_\zeta}{\text{number of used timeslots}} \tag{9}$$

C. Exploration Rate

In this paper we investigate two variants for exploring the environment; Greedy epsilon and Epsilon decay.

1) Greedy epsilon: we fix the epsilon parameter over all steps for all episodes to explore and exploit the environment. Accordingly, an agent selects a random action with a probability equal to ϵ.

2) Epsilon decay: the epsilon is initially set to 1, meaning that the agent chooses actions at random to explore the new environment and update the Q-table. After each episode, the epsilon's value decays, to shrink the exploration rate as the Q-table's values are refined through several steps, hence, the agent starts exploiting the learned experience.

V. Evaluation

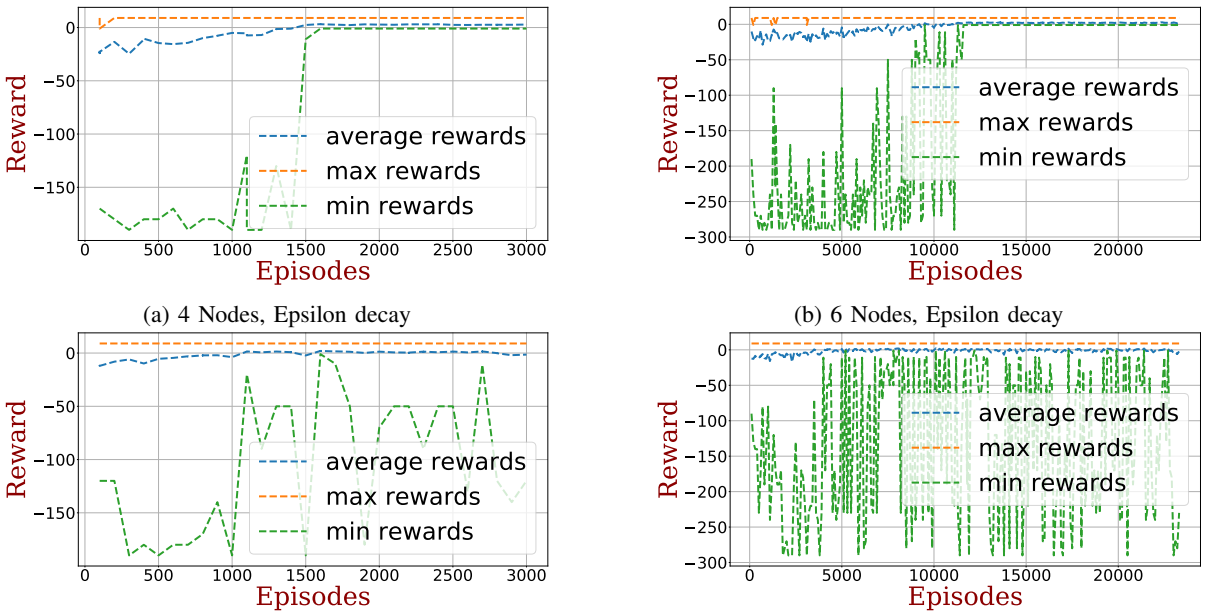
We define two different infrastructure graphs with 4 and 6 nodes. For each one we change the source and destination nodes, the attenuation between the nodes, and the nodes' capacities, to have in total 50 different scenario per each graph. We fix the overlay graph to a linear chain of five processing blocks, where all links in the overlay graph require the same data rate.

We set the hyper-parameters for the RL algorithm to $\alpha = 0.1$ and $\gamma = 0.6$. When using Greedy epsilon we set $\epsilon = 0.1, 0.5, 0.9$ and fix it later to $\epsilon = 0.1$ (Section V-A). For the Epsilon decay variant, we decrease the value of ϵ by $\frac{1}{\frac{|\text{episodes}|}{2}-1}$, and stop decaying the epsilon after $\frac{|\text{episodes}|}{2}$ episodes. We set the maximum number of episodes to triple the number of state, so that we have 3072 episodes and 23328 episodes for 4- and 6-nodes networks, respectively.

A. Value based convergence

We first investigate the Q-table's convergence when using Epsilon decay and Greedy epsilon with different values for ϵ (0.1, 0.5 and 0.9). In Figure 4, we observe that when using $\epsilon = 0.1$ or using Epsilon decay, the mean value for the Q-table seems to converge within 2000 episodes for 4 nodes network (Figure 4a) and 10000 episodes for 6 nodes network (Figure 4b). However, when $\epsilon = 0.5$ or 0.9, the mean value does not converge for 4 nodes network nor for 6 nodes network.

It is expected that the higher the value for ϵ it is, the more the number of episodes to converge it needs. Nevertheless, if we choose a very low value for ϵ, we will converge fast but also prematurely to a misleading value, because we do not explore all possible good choices. Accordingly, we fix Greedy epsilon to $\epsilon = 0.1$ for the rest of the evaluation, and compare the goodness in Section V-C.



(a) 4 Nodes, Epsilon decay

(b) 6 Nodes, Epsilon decay

(c) 4 Nodes, Greedy epsilon

(d) 6 Nodes, Greedy epsilon

Fig. 6: Average reward over episodes.

B. Exploration and Exploitation

To investigate the exploration for Greedy epsilon and Epsilon decay, we show in Figure 5 the resulted maximum reward from all actions for each state. The baseline represents the maximum ground truth for each state. For 4 nodes (Figure 5a), we have 1024 states, where both Greedy epsilon and Epsilon decay tend to give higher weights for the lower numbered states. We observe a similar behavior for 6 nodes network (Figure 5b) with 7776 states.

This behavior is due to the greediness of both approaches; the baseline has the same reward for different states, therefore, a greedy approach will select an action that exploits the first (i.e., the lowest number) state, out of all possible actions with the same reward.

We take a deeper look at the reward and see how it evolves with increasing the number of episodes (Figure 6); over the last 100 episodes, we show the maximum, minimum and average rewards for the selected actions.

On the one hand, we observe that for the Epsilon decay method (Figure 6a and Figure 6b), the minimum rewards converge and become close to the average reward after 1600 and 11000 episodes for 4 and 6 nodes networks, respectively. This is due to the decaying property of $epsilon$, where the agent becomes greedier as $epsilon$ approaches zero.

On the other hand, the minimum reward for the Greedy epsilon (Figure 6c and Figure 6d) has high variance, because the greediness of the agent is fixed ($\epsilon = 0.1$) and it does not always aim at maximizing the reward, but also try other random actions. Meanwhile, the average values converge similar to the Epsilon decay method.

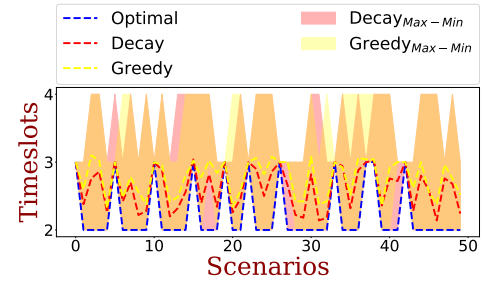
C. Comparison to the Optimization Model

We observe the resulted timeslots using Greedy epsilon and Epsilon decay (Figure 7). We simulate 50 different scenario, where each scenario has different source and sink nodes, different attenuation between the nodes, and different capacities per nodes. Each scenario is solved 50 different times using Greedy epsilon and Epsilon decay methods. Out of the 50 solutions, we calculate the mean, maximum and minimum achieved number of timeslots. Additionally, we calculate the optimal number of timeslots per each scenario using Pyomo interface [1] and Gurobi solver [2].

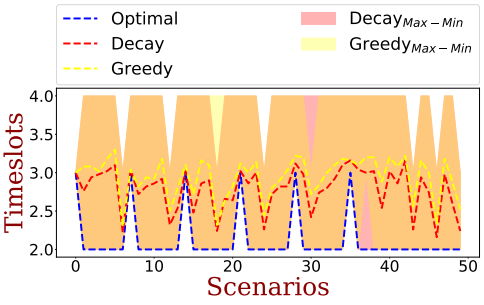
As expected, the optimal number of timeslots (blue) act as lower bounds for both Epsilon decay (red) and Greedy epsilon (yellow) methods. Moreover, we observe that in worst case scenarios, for both methods, the additional number of timeslots (compared to the optimal solution) is 2 timeslots, for both 4- (Figure 7a) and 6-nodes (Figure 7b) networks. Meanwhile, the mean number of timeslot of the Epsilon decay is always less than or equal to that of Greedy epsilon.

We show in Figure 8 the confidence interval for the average number of timeslots using both methods for all scenarios per each network. Although the Epsilon decay method has lower mean for both networks, the confidence interval cannot really conclude that the decay method is always better than the greedy one. Nevertheless, the results from Figure 7a and Figure 7b show that the the decay method is at least as good as the greedy one in all 50 scenarios. It is worth mentioning the number of steps required to find a valid solution are almost

[1] www.pyomo.org
[2] www.gurobi.com



(a) 4 Nodes



(b) 6 Nodes

Fig. 7: Resulted number of timeslots

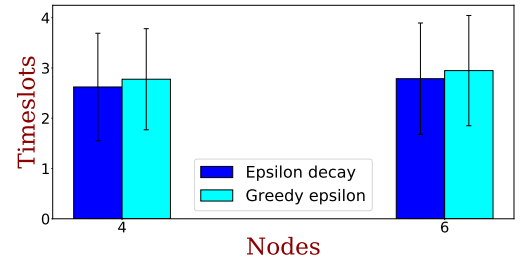


Fig. 8: Epsilon decay vs Greedy epsilon

the same: on average 1.8 steps using the Epsilon decay and 1.98 steps using the Greedy epsilon.

VI. Conclusion

In this paper, Greedy epsilon and Epsilon decay methods have been used for exploration and the results are compared to the optimization model.

A major benefit of the RL-trained solutions is that they are 1000 times faster than the optimization model. We show that the proposed framework results in solutions that have on average one additional timeslot for transmission compared to the optimization model. Although both exploration methods seem to have similar performance, the Epsilon Decay tends to yield a better solution: faster and closer to the optimal.

References

[1] A. Nelus and R. Martin, "Gender discrimination versus speaker identification through privacy-aware adversarial feature extraction," in *Speech Communication; 13th ITG-Symposium*, Oct 2018, pp. 1–5.

[2] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A comprehensive survey on fog computing: State-of-the-art and research challenges," *IEEE Communications Surveys Tutorials*, vol. 20, no. 1, pp. 416–464, Firstquarter 2018.

[3] R. Riggio, A. Bradai, D. Harutyunyan, T. Rasheed, and T. Ahmed, "Scheduling wireless virtual networks functions," *IEEE Transactions on Network and Service Management*, vol. 13, no. 2, pp. 240–252, June 2016.

[4] Vodafone, "Vodafone dreamlab," 2020. [Online]. Available: https://www.vodafone.com/about/vodafone-foundation/focus-areas/dreamlab-app

[5] J. Ebbers, J. Heitkaemper, J. Schmalenstroeer, and R. Haeb-Umbach, "Benchmarking neural network architectures for acoustic sensor networks," in *Speech Communication; 13th ITG-Symposium*, Oct 2018, pp. 1–5.

[6] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[7] M. Zargham, *Ch.3, Computer Architecture: Single and Parallel Systems*, ser. Prentice Hall International Series in. Prentice Hall, 1996.

[8] J. van de Belt, H. Ahmadi, and L. E. Doyle, "Defining and surveying wireless link virtualization and wireless network virtualization," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1603–1627, thirdquarter 2017.

[9] H. Afifi, S. Auroux, and H. Karl, "MARVELO: wireless virtual network embedding for overlay graphs with loops," in *2018 IEEE Wireless Communications and Networking Conference (WCNC) (IEEE WCNC 2018)*, Apr. 2018.

[10] L. Chen, S. Abdellatif, A. SimoTegueu, and T. Gayraud, "Embedding and re-embedding of virtual links in software-defined multi-radio multi-channel multi-hop wireless networks," *Computer Communications*, vol. 145, pp. 161 – 175, 2019.

[11] H. Afifi and H. Karl, "An approximate power control algorithm for a multi-cast wireless virtual network embedding," in *2019 12th IFIP Wireless and Mobile Networking Conference (WMNC)*, Sep. 2019, pp. 95–102.

[12] V. Cionca, R. Marfievici, R. Katona, and D. Pesch, "JudiShare: Judicious resource allocation for QoS-based services in shared wireless sensor networks," in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, April 2018, pp. 1–6.

[13] H. Afifi, K. Horbach, and H. Karl, "A genetic algorithm framework for solving wireless virtual network embedding," in *2019 International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Oct 2019, pp. 1–6.

[14] X. Gao, W. Zhong, Z. Ye, Y. Zhao, J. Fan, X. Cao, H. Yu, and C. Qiao, "Virtual network mapping for reliable multicast services with max-min fairness," in *2015 IEEE Global Communications Conference (GLOBECOM)*, Dec 2015, pp. 1–6.

[15] L. Chen, S. Abdellatif, A. F. Simo, and T. Gayraud, "Virtual link embedding in software-defined multi-radio multi-channel multi-hop wireless networks," in *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, ser. MSWIM 18. New York, NY, USA: Association for Computing Machinery, 2018, p. 163172.

[16] S. Abdelwahab, B. Hamdaoui, M. Guizani, and T. Znati, "Efficient virtual network embedding with backtrack avoidance for dynamic wireless networks," *IEEE Transactions on Wireless Communications*, vol. 15, no. 4, pp. 2669–2683, April 2016.

[17] X. Guo, H. Lin, Z. Li, and M. Peng, "Deep reinforcement learning based qos-aware secure routing for sdn-iot," *IEEE Internet of Things Journal*, pp. 1–1, 2019.

[18] X. Fu, F. R. Yu, J. Wang, Q. Qi, and J. Liao, "Dynamic service function chain embedding for nfv-enabled iot: A deep reinforcement learning approach," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 507–519, Jan 2020.

[19] J. Wang, L. Zhao, J. Liu, and N. Kato, "Smart resource allocation for mobile edge computing: A deep reinforcement learning approach," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2019.

[20] H. Afifi and H. Karl, "RL Framework for VNE," 2020. [Online]. Available: https://git.cs.upb.de/hafifi/rl-for-vne.git

[21] M. Wiering and M. van Otterlo, *Reinforcement Learning: State-of-the-Art*, ser. Adaptation, Learning, and Optimization. Springer Berlin Heidelberg, 2012.

[22] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018.