Learn to improve: A novel deep reinforcement learning approach for beyond 5G network slicing

Anouar Rkhami, Yassine Hadjadj-Aoul, Abdelkader Outtagarts

Learn to improve: A novel deep reinforcement learning approach for beyond 5G network slicing

Anouar Rkhami
Inria, Univ. Rennes, CNRS, IRISA

Yassine Hadjadj-Aoul
Univ. Rennes, Inria, CNRS, IRISA

Abdelkader Outtagarts
Nokia-Bell Labs, France

***Abstract*—Network slicing remains one of the key technologies in 5G and beyond 5G networks (B5G). By leveraging SDN and NVF techniques, it enables the coexistence of several heterogeneous virtual networks (VNs) on top of the same physical infrastructure. Despite the advantages it brings to network operators, network slicing raises a major challenge: Resource allocation of VNs, also known as the virtual network embedding problem (VNEP). VNEP is known to be an NP-Hard problem. Several heuristics, meta-heuristics and Deep Reinforcement Learning (DRL) based solutions were proposed in the literature to solve it. Regarding the first two categories, they can provide a solution for large scale problems within a reasonable time, but the solution is usually suboptimal, which leads to an inefficient utilization of the resources and increases the cost of the allocation process. For DRL-based approaches and due to the exploration-exploitation dilemma, the solution can be infeasible. To overcome these issues, we combine, in this work, deep reinforcement learning and relational graph convolutional neural networks in order to automatically learn how to improve the quality of VNEP heuristics. Simulation results show the effectiveness of our approach. Starting with an initial solution given by the heuristics our approach can find an amelioration, with an improvement in the order of 35%.**

***Index Terms*—5G , Virtual Network Embedding, Deep Reinforcement Learning, Relational Graph Convolutional Neural Networks, Resource Allocation**

I. Introduction

5G networks are expected to handle multiple services with different types of requirements within a single physical infrastructure. In order to achieve such an objective, the Network Slicing (NS) [1] is certainly one of the most important building blocks.

NS leverages the last advents of Network Function Virtualization [11](NFV) and Software Defined Networking [10] (SDN) to enable the coexistence of heterogeneous Virtual Network Requests (VNRs) on top of the same Substrate Network (SN). In addition, NS provides network infrastructure providers (InPs) with new opportunities to increase their revenues. In fact, they will be more involved in the resource-provisioning process by leasing their infrastructure to third parties.

Despite the benefits it brings, NS raises a plethora of challenges. The allocation of resources for network slices is considered to be one of the most important issues, which is known in the literature as the Virtual Network Embedding Problem [2] (VNEP). VNEP is known to be an NP-Hard problem [6] and several families of solutions have been proposed to solve it: Exact, heuristics, metaheuristics and finally deep reinforcement-based approaches [13].

While exact solutions ensure optimality, they are applicable only to small network instances [7]. For large networks, heuristics may find a solution within a reasonable time-frame, but they are sub-optimal [5]. To avoid this issue, meta-heuristics were proposed, they bring more flexibility and more efficiency in the exploration process [12], but they lack efficiency in the solution computation process. Indeed, they don't accumulate knowledge, for every new instance a new search process must be started from scratch. More recently, some works unveiled the potential of deep reinforcement learning to solve the VNEP [14]. This category of approaches is more promising since with reinforcement learning we can overcome the issues stated before and learn from past experiences and with deep learning techniques we can extract the most relevant features automatically. However, due to the exploration-exploitation dilemma, the learning may lead to choose infeasible solution which may cause the rejection of critical network services as well as a decrease in revenues for InPs.

In this paper, we propose a new way of using DRL to solve the VNEP. Instead of learning a solution from scratch, which may result in an unsafe learning, we propose to train an agent to reduce the optimality gap of VNEP heuristics. This will lead to reliable solutions based on the DRL. Also, this will ensure that the worst-case result is equal to the one obtained with the heuristic.

To achieve this objective, we model the process of improving the quality of the heuristics as a reinforcement learning problem, where we train an agent to find the best strategy (policy). In each iteration of the training, the RL agent tries to improve sequentially the quality of the placement of the slices given by a heuristic.

The main contributions of this paper can be summarized as follows :

- We propose a safe DRL-based solution to the problem of resources allocation for network slices by learning how to reduce the optimality gap of heuristic-based solutions,
- We formalize the task as a Markov Decision Process (MDP),
- We represent the states of the system in a novel way using heterogeneous graphs,
- In order to automate the features extraction process and let the agent benefit from the structural information, we

model its policy using a Relational Graph Convolutional Neural-based architecture,
- We design a specific reward function in order to guide the agent to improve the quality of the solutions.

The remaining of this paper is structured as follows. The formulation of the VNEP problem is presented in section III. Then, the MDP formulation of the problem is presented in section IV. The neural network architecture and the training procedure is depicted in section V. The simulation setup and the performance evaluation are presented in section VI and the conclusions of the work are provided in section VII.

II. Background

A. Heterogeneous Graphs

A heterogeneous graph and heterograph for short [17], is defined as a directed graph $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \mathcal{T}_n, \mathcal{T}_l)$ associated with two functions ϕ_n and ϕ_l. $\mathcal{V}$, $\mathcal{E}$, $\mathcal{T}_n$, $\mathcal{T}_l$ represent, respectively, the set of nodes, the set of edges, the set of types of nodes and the set of types of links of the heterograph. $\phi_n \colon \mathcal{V} \rightarrow \mathcal{T}_n$ maps each node $v \in \mathcal{V}$ to a single node type $\phi_n(v)$, while $\phi_l \colon \mathcal{E} \rightarrow \mathcal{T}_l$ maps each link $e \in \mathcal{E}$ to a link type $\phi_l(e)$.

Besides, $\mathcal{T}_n$ and $\mathcal{T}_l$ satisfy the following condition:

$$|\mathcal{T}_n| + |\mathcal{T}_l| > 2 \tag{1}$$

B. Relational Graph Convolutional Neural Networks (RGCN)

Graph Convolutional neural Networks(GCN) [9] were proposed as a generalization of the convolution operation on arbitrary graph-structured data. The aim of GCN is to extract features automatically from graphs. However, GCN can deal only with homogeneous graphs where we have only one type of edges between nodes (i.e $|\mathcal{T}_l| = 1$). Hence, RGCN [16] were defined as an extension of GCN to extract features from heterographs.

The key idea behind RGCN is that the encoding of a node is based on the encoding of its neighbors under all types of links.

Let $\mathcal{H}$=($\mathcal{V}$,$\mathcal{E}$,$\mathcal{T}_n$,$\mathcal{T}_l$) be a heterograph. We assume that each node $v \in \mathcal{V}$ has an input vector in_v. Each layer of an RGCN network takes as input the feature vector in_v and outputs a new vector out_v according to the following equation :

$$out_v = \sigma(\sum_{r \in \mathcal{T}_l} \sum_{j \in \mathcal{N}_v^r} \frac{1}{c_{v,r}} W_r in_v) \tag{2}$$

where $\mathcal{N}_v^r$ represents the neighbors of node v under the link type r, $c_{v,r} = |\mathcal{N}_v^r|$ represents a normalization constant and finally W_r denotes the trainable weight matrix associated with the link type r.

III. Problem Formulation

In this section, we describe the model of the SN, the VNR and their resources. Then, we define formally the VNE problem and the metrics to optimize, and by which we quantify the quality of the placement. The notations used in this section are summarized in Table I.

A. Networks and resources models

The susbtrate network is defined by an undirected graph $\mathcal{G}^s = (\mathbb{N}^s, \mathbb{L}^s)$, where $\mathbb{N}^s$ represents the set of substrate nodes and $\mathbb{L}^s$ the set of substrate links. The number of the substrate nodes and substrate links are $|\mathbb{N}^s|$ and $|\mathbb{L}^s|$, respectively. Each substrate node n^s has a computing capacity c_{n^s} and each substrate link l^s has a bandwidth capacity b_{l^s}. Similarly, the VNR is represented by a directed graph $\mathcal{G}^v$ = $(\mathbb{N}^v, \mathbb{L}^v)$, where $\mathbb{N}^v$ represents the set of VNFs and $\mathbb{L}^v$ represents the set of virtual links. The numbers of VNFs and VLs are $|\mathbb{N}^v|$ and $|\mathbb{L}^v|$, respectively. Each VNF n^v has a computing demand denoted as c_{n^v}, and each VL l^v has a bandwidth demand b_{l^v}.

Notation	Description
$\mathcal{G}^s, \mathcal{G}^v$	The substrate network and VNR graphs
$\mathbb{N}^s, \mathbb{N}^v$	Set of substrate nodes and VNFs
$\mathbb{L}^s, \mathbb{L}^v$	Set of substrate links and VLs
$\mathbb{P}^s$	Set of substrate paths
c_{n^s}	Available CPU at substrate node n^s
b_{l^s}	Available bandwidth of substrate link l_s
c_{n^v}	CPU request of VNF n^v
b_{l^v}	Bandwidth request of VL l_v

TABLE I: Notations

B. VNE Problem

The VNE problem can be divided into two stages: the virtual node mapping VNM and the virtual link mapping VLM. The former is defined as a function $f_{\text{VNM}} \colon \mathbb{N}^v \rightarrow \mathbb{N}^s$ that maps a virtual node to a substrate node. It must be injective, that is, two VNFs of the same VNR cannot be hosted by the same substrate network.

$$f_{\text{VNM}}(v_i) = f_{\text{VNM}}(v_j) \Rightarrow v_i = v_j \forall v_i, v_j \in \mathbb{N}^v \tag{3}$$

On the other hand, VLM can be modeled as a function $f_{\text{VLM}} \colon \mathbb{L}^v \rightarrow \mathbb{P}^s$ that maps a virtual link to a physical path (set of physical links).

At the node mapping level, a virtual node is successfully deployed if the substrate node that hosts it has sufficient CPU resources,

$$c_{n^v} \leq c_{f_{\text{VNM}}(n^v)}, \quad \forall n^v \in \mathbb{N}^v. \tag{4}$$

At the link mapping stage, a VL is successfully deployed if its virtual nodes are deployed and if each physical link belonging to the physical path on which it is mapped has sufficient bandwidth,

$$b_{l^v} \leq \min_{l^s \in f_{vlm}(l^v)} b_{f_{\text{VLM}}(l^v))}, \quad \forall l^v \in \mathbb{L}^v. \tag{5}$$

C. Optimization objective

In this work, the aim is to find a placement strategy of VNRs that uses efficiently the substrate network resources and maximize the InPs revenue. To quantify the quality of the embedding, several works used the revenue generated by embedding a request R as well as its cost C [4]. The revenue

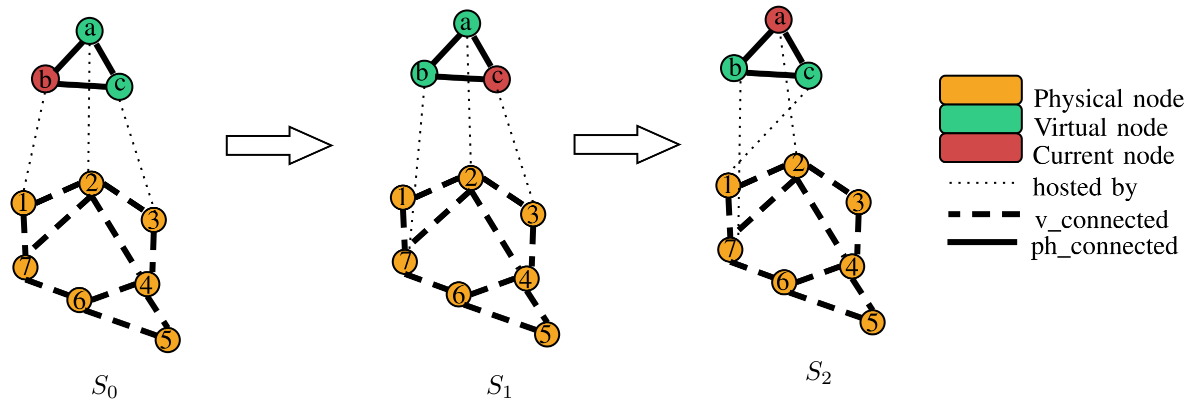

Fig. 1: Illustration of the sequential process of Improvement of quality of VNE heuristics. The state is represented using an heterograph whit two types of nodes and three types of links. At each step either the placement of a virtual node is modified or kept.

and the cost of embedding a VNR $\mathcal{G}^v$are defined respectively as follows :

$$R(\mathcal{G}^v) = \sum_{n^v \in \mathbb{N}^v} c_{n^v} + \sum_{l^v \in \mathbb{L}^v} b_{l^v} \tag{6}$$

$$C(\mathcal{G}^v) = \sum_{n^v \in \mathbb{N}^v} c_{n^v} + \sum_{l^v \in \mathbb{L}^v} b_{l^v} \times |\mathbb{P}^s_{l^v}| \tag{7}$$

where $|\mathbb{P}^s_{l^v}|$ represents the length of the physical path that hosts the virtual link l_v. Since we want both: reducing the cost and increasing the revenue, we consider a new metric called Revenue to Cost (R2C), which is defined as follows :

$$R2C(\mathcal{G}^v) = \begin{cases} \frac{R(\mathcal{G}^v)}{C(\mathcal{G}^v)}, \text{ if } \mathcal{G}^v \text{ is accepted} \\ 0 \text{ otherwise} \end{cases}$$

A VNR is accepted if, and only if, all its nodes and links mapping satisfy, respectively, the constraints (4) and (5).

IV. MDP FORMULATION

Instead of using DRL to find a solution for the VNEP, we will adopt it in order to optimize the quality of a solution given by a heuristic. In this section we give an overview of DRL and Markov Decision Process (MDP). Then we define the problem of Improving the Quality of VNEP Heuristics (IQH) within this framework.

A. Deep Reinforcement learning overview

In reinforcement learning [15], we have two main entities, an environment and an agent. The learning process is done through the interaction between them so that the agent can optimize a total amount of return. At every time step t the agent gets a state representation s_t from the environment and chooses an action a_t based on the state representation. Then, the agent applies this action on the environment. As result the environment moves into a new state s_{t+1} and the agent receives a reward r_t corresponding to this transition as well as the representation of the new state. This interaction can be modeled as an MDP $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ with a state space $\mathcal{S}$, an action space $\mathcal{A}$, transition dynamics $\mathcal{P}$ and a reward function $\mathcal{R}$. The behavior of the agent is defined by a policy function $\pi\colon \mathcal{S} \rightarrow \mathcal{A}$ that maps the state to a distribution over the actions. The aim of RL is to find the optimal policy π^* that maximizes the expected cumulative return. To achieve this, traditional RL methods used tabular representations for π. However, these approaches reached their limits when the state and action spaces are high. To deal with this, Deep Neural Networks (DNNs) were used instead of tabular methods to approximate the policy function. Moreover, DNNs help the agent to extract the most relevant features from the state representation.

B. Improving the Quality of VNEP Heuristics (IQH) as MDP

Given a VNE problem and an heuristic to solve it, the process of improving the quality of heuristics consists in starting with this initial solution and then modify it in order to increase its R2C score. Instead of modifying the solution in a one shot basis, we do it node by node. We define IQH as a sequential MDP where the states, actions and rewards are defined as follows:

- **States:** In RL, the state represents the raw information the agent gets from the environment and based on which it chooses an action. In this work we define the state as a hetero-graph that illustrates the solution of the heuristic. The hetero-graph is composed of 2 types of nodes and 3 types of edges : ‘v_connected’, ‘hosted_by’ and ‘ph_connected’. The two categories of nodes represent the virtual and substrate nodes, while the categories of edges represent the type of possible relation between these nodes. Each virtual node is connected to another virtual node under the relation *v_connected* and each substrate node is connected to another substrate node under the relation *s_connected*, and to represent the solution given by the heuristic , each VNF may be connected to a substrate node via *hosted_by* type relation Each node has 3 initial features. For the virtual nodes: (1) the CPU required by the VNF c_{n^v}, (2) the sum of requested bandwidth defined as the total bandwidth requested by the virtual links to which belongs the node; and (3) a flag indicating if the VNF is the current VNF to process. On the other hand each substrate node has the following features : (1) the remaining amount of CPU c_{n^s}, (2) the sum of bandwidth defined as the total remaining bandwidth of links to which belongs the substrate node; and (3) the number of its neighbors. Initially, a random VNF (virtual node) is selected as the current node.

- **Actions:** At each time step t, the agent has to select either to keep the same placement of the current VNF or to modify it into another substrate that does not host any other VNF from the same request. Hence the action space at time step t is defined as follows :

$$\mathcal{A}_t = \mathbb{N}^s \backslash \mathbb{S}_t \tag{8}$$

where $\mathbb{S}_t$ represents the set of substrate nodes that host a VNF of the current slice at time step t excepting the current node. For example, as depicted in Figure **??** at step 0, $\mathcal{A}_0 = \{1, 7, 6, 4, 5\}$.
Once the action for the current node is computed, we check its feasibility (i.e., chosen node has enough resources) and then we try updating the link mapping of edges, where the current VNF is head or tail. We run the shortest path algorithm between the new chosen substrate node and the placement of its neighbors under the relation '*connected_v*'. If the link mapping is feasible the agent will receive a new state representation, where the '*hosted_by*' link of the current VNF is updated according to the new computed node and a new virtual node is marked as the current node. If a failure occurs either on the node or on the link mapping stages, the action is not applied and the agent gets a new state representation, where only the current VNF is changed. Besides, after each step the agent gets a reward related to the action it made.
- **Rewards:** The reward is the signal by which a RL agent can measure the correctness of the action it made. It can be positive in the successful cases or negative if the action leads to an undesirable behavior. The process to compute the reward obtained at each time step t is presented in Algorithm **??**. The objective is to improve the R2C score of the solution given by the heuristic by updating the placement of virtual nodes sequentially, hence at each time step t if the action a_t leads to an unfeasible solution the agent gets a negative reward of -100 and we keep the last feasible placement. Otherwise, the reward obtained is $r_t = rc_t - bp_t$, where rc_t is the new R2C score after updating the placement of the t^{th} node and bp_t is the best R2C score found before the time step t. Initially bp_0 is the R2C score of the placement of the heuristic.

V. Agent's design

In this section we present the neural network-based architecture of the policy as well as its training process.

Algorithm 1 Compute reward

```
1:  function getReward(bp, r2c)
2:      reward ← 0
3:      if r2c = 0 then
4:          reward ← −100                ▷ unfeasable solution
5:      else
6:          reward ← (r2c − bp)
7:      end if
8:      if r2c > bp then
9:          bp ← r2c                     ▷ new best score
10:     end if
11:     return reward, bp
12: end function
```

A. Agent Architecture

To choose actions, the policy of the agent is parametrized using deep neural networks and trained using the policy gradient descent algorithm.

The architecture of the agent is depicted in the Figure 2. The policy is a function that maps each state representation to a probability distribution over the actions. At each time step t given the state representation, the policy network outputs a distribution probability over the actions. However, since the state is represented by a heterograph, the first part of the policy network is in charge of encoding this information. This step generates a vector representation of all nodes in the heterograph using an RGCN network, then the vector of the current virtual node is selected and fed to a Multi-Layer Perceptron (MLP)

B. Training

To optimize the neural network architecture of the policy network and update its trainable parameters, we adopt the REINFORCE [19] algorithm. REINFORCE is a policy-gradient algorithm, in other words it updates the policy network parameters to optimize an objective function using gradient descent. The objective is to maximize the cumulative discounted return:

$$R_t = \mathbb{E}[\sum_t \gamma^t r_t], \quad \gamma \in [0, 1]. \tag{9}$$

where γ is the discount factor, by which we ensure the convergence of the cumulative return in the infinite horizon setups. The gradient of this objective is defined by :

$$\nabla R_t = \mathbb{E}[\log(\pi_\theta(a_t|s_t))Q_{\pi_\theta}(a_t, s_t)] \tag{10}$$

where $Q_{\pi_\theta}(a_t, s_t)$ represents the expected cumulative reward by choosing a_t in s_t and following π_θ. θ represents the set of trainable parameters of π
With Reinforce, this gradient is estimated using a MONTE-CARLO approach. During each episode a rollout (s_t, a_t, r_t) is performed using the current policy π_θ. At the end of each episode a gradient ascent step is performed and θ are updated as follows :

$$\theta \leftarrow \theta + \sum_{t=0}^{T} \alpha \log(\pi_\theta(a_t|s_t))(\sum_{k=t}^{T} r_k - b_k) \tag{11}$$

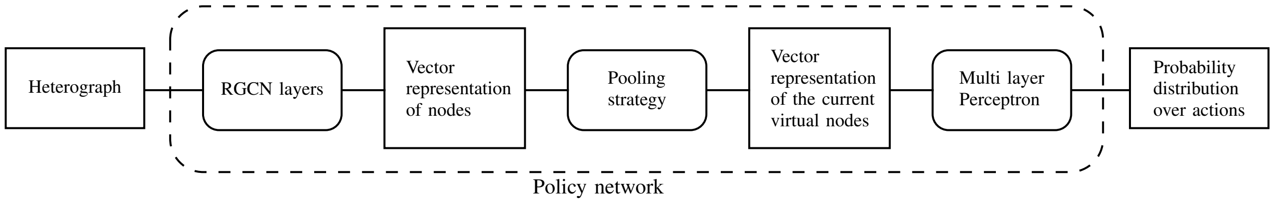
Heterograph
RGCN layers
Vector representation of nodes
Pooling strategy
Vector representation of the current virtual nodes
Multi layer Perceptron
Probability distribution over actions
Policy network

Fig. 2: The neural architecture of the policy function.

Parameters	Values
Learning rate α	5×10^{-3}
Discount Factor γ	0.99
Optimizer	Adam [8]

TABLE II: DRL agent Hyperparameters

b_t represents a baseline for reducing the variance of the estimate and it is equal to the average of the cumulative reward starting from time step t, $b_t = \sum_{k=t}^{T} r_k - b_k$. This way, the trainable parameters are updated such that the probability of actions leading to a cumulative rewards higher than the average rewards is increased.

VI. Performance evaluation

In this section, we present the simulation setup and an analysis of the obtained results.

A. Simulation setup

To assess the performance of our approach we consider the following simulation setup.

For the substrate network, we consider a random topology generated following the Waxman random graph model [18] , using the parameters α = 0.5 and β = 0.2; this methodology for topology generation has been commonly used in previous works [4]. Given this configuration, the generated substrate network will contain 100 nodes and 500 edges. The CPU and bandwidth of the substrate nodes and links are drawn uniformly from the interval [50,100].

To generate the virtual requests topology, we adopt the Erdős–Rényi model [3]. The generated topology is defined by the numbers of nodes n and the probability p for the edges creation. With $p = 2\frac{\ln(n)}{n}$, the generated graphs are in general connected. The CPU and bandwidth requested by the VNR are drawn uniformly from the interval [10,20]. The number of VNFs in each VNR is randomly chosen from the interval [8,10].

For the DRL agent, the policy architecture is set up with the parameters depicted in Table II. The RGCN part is represented by a 2-layer RGCN network. The first layer contains 16 hidden units and the second one 32 hidden units. The fully connected layer that maps the vector encoding the current node to the actions contains 100 units. We adopt a LeakyReLu activation for all layers except for the last one where we used softmax activation so as to get probabilities.

Each time a VNR arrives, the heuristic finds first the initial placement, and then an episode of quality improvement starts. There are 30000 episodes. We execute 10 runs with different seeds.

The model architecture , is written in `Python` using the `Pytorch`[1] and the `DGL`[2] libraries.

B. Heuristics

We evaluated our method in learning to improve the following heuristics :

- **First-Fit**: An heuristic that maps a virtual node into the first substrate node with sufficient CPU.
- **Best-Fit**: Heuristic that maps a virtual node to the substrate node with the smallest sufficient CPU.

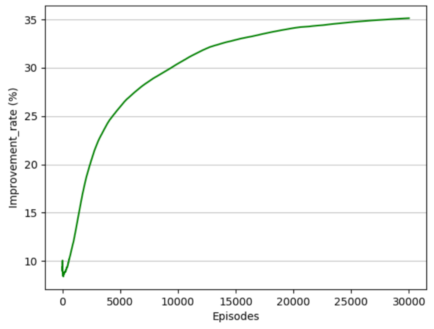

Fig. 3: Improvement rate while using First Fit.

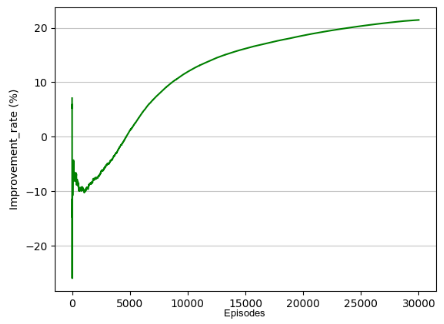

Fig. 4: Improvement rate while using Best Fit.

Figures 3 and 4 show, respectively, the improvement rates reached by the agent, for First-Fit and Best-Fit. For the First-Fit, we see that the agent could reach an improvement rate of

[1]https://pytorch.org/
[2]https://www.dgl.ai/

35%. Moreover, the performance of the agent gets better with time. On the other hand, with the Best-Fit strategy, the task was harder for the agent in the beginning, but with the negative rewards it got, the agent learned to avoid bad decisions and converges to an improvement rate of 20%. Note that when the agent fails to improve the heuristic solution (the worst case) we don't apply its decision but it learns from it.

To assess the robustness of the proposed approach we modify the interval from which the number of VNFs per VNR is drawn from [8,10] to [12,14]. Figures 6 and 5 show the results using the First-Fit and the Best-Fit strategies under this setting. Even the task becomes harder, the agent learns and finds a way to improve both heuristics under this new setting. For First-Fit the agent reached an average improvement rate of 16%, while for Best-Fit the improvement rate was of 13%.

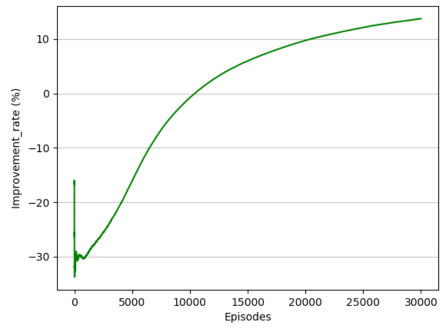

Fig. 5: Improvement rate while using Best Fit .

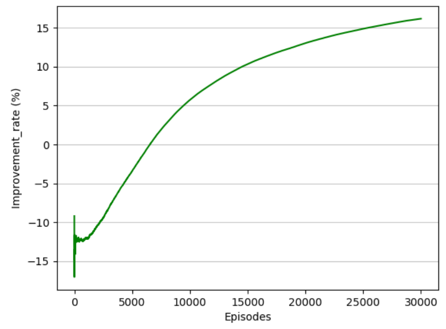

Fig. 6: Improvement rate while using First Fit.

VII. Conclusions

Virtual network embedding in 5G and beyond 5G networks is known to be an NP-hard problem (when properly converted to a decision problem). In this work we unveiled the potential of deep reinforcement learning in a novel way. Instead of learning a solution from scratch using DRL we trained a reinforcement learning agent to reduce the optimality gap of heuristics dedicated to solve this problem. We modeled the task as a sequential Markov decision process, where the states are represented using heterogeneous graphs. To exploit the graph information we exploited RGCN to parameterize the policy network and in order to guide the agent to the desired behavior we defined a reward function that gives the agent a positive reward each time it could improve the quality of the heuristic. To validate our approach we learned to improve two heuristics: First-Fit and Best-Fit. the simulation results showed that we can reach an improvement rate of 35% for First-Fit and 20% for Best-Fit.

In the future, we are planning to learn to improve other heuristics and under different settings.

References

[1] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck. Network slicing and softwarization: A survey on principles, enabling technologies, and solutions. *IEEE Communications Surveys Tutorials*, 20(3):2429–2453, 2018.

[2] H. Cao, S. Wu, Y. Hu, Y. Liu, and L. Yang. A survey of embedding algorithm for virtual network embedding. *China Communications*, 16(12):1–33, 2019.

[3] P. Erdős and A. Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[4] A. Fischer. An evaluation methodology for virtual network embedding. 2017.

[5] L. Gong, Y. Wen, Z. Zhu, and T. Lee. Toward profit-seeking virtual network embedding algorithm via global resource capacity. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pages 1–9, 2014.

[6] A. Gupta, B. Jaumard, M. Tornatore, and B. Mukherjee. A Scalable Approach for Service Chain Mapping With Multiple SC Instances in a Wide-Area Network. *IEEE Journal on Selected Areas in Communications*, 36(3):529–541, March 2018.

[7] I. Jang, D. Suh, S. Pack, and G. Dán. Joint optimization of service function placement and flow distribution for service function chaining. *IEEE Journal on Selected Areas in Communications*, 35(11):2532–2541, 2017.

[8] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[9] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[10] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

[11] Y. Li and M. Chen. Software-defined network function virtualization: A survey. *IEEE Access*, 3:2542–2553, 2015.

[12] T. A. Q. Pham, J.-M. Sanner, C. Morin, and Y. Hadjadj-Aoul. Virtual network function–forwarding graph embedding: A genetic algorithm approach. *International Journal of Communication Systems*, 33(10):e4098, 2020.

[13] P. T. A. Quang, Y. Hadjadj-Aoul, and A. Outtagarts. A deep reinforcement learning approach for vnf forwarding graph embedding. *IEEE Transactions on Network and Service Management*, 16(4):1318–1331, 2019.

[14] A. Rkhami, P. Tran Anh Quang, Y. Hadjadj-Aoul, A. Outtagarts, and G. Rubino. On the use of graph neural networks for virtual network embedding. In *2020 International Symposium on Networks, Computers and Communications (ISNCC)*, 2020.

[15] S. J. Russell and P. Norvig. *Artificial intelligence: a modern approach.* Malaysia; Pearson Education Limited,, 2016.

[16] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer, 2018.

[17] C. Shi, Y. Li, J. Zhang, Y. Sun, and S. Y. Philip. A survey of heterogeneous information network analysis. *IEEE Transactions on Knowledge and Data Engineering*, 29(1):17–37, 2016.

[18] B. M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, 1988.

[19] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256, May 1992.