A Robust Monte-Carlo-Based Deep Learning Strategy for Virtual Network Embedding

Ghina Dandachi, Anouar Rkhami, Yassine Hadjadj-Aoul, Abdelkader Outtagarts

A Robust Monte-Carlo-Based Deep Learning Strategy for Virtual Network Embedding

Ghina Dandachi[1], Anouar Rkhami[1], Yassine Hadjadj-Aoul[1], and Abdelkader Outtagarts[2]

[1]*Inria, Univ Rennes, CNRS, IRISA, France*
[2]*Nokia-Bell Labs, France*

***Abstract*—Network slicing is one of the building blocks in Zero Touch Networks. It mainly consists in a dynamic deployment of services in a substrate network. However, the Virtual Network Embedding (VNE) algorithms used generally follow a static mechanism, which results in sub-optimal embedding strategies and less robust decisions. Some reinforcement learning algorithms have been conceived for a dynamic decision, while being time-costly. In this paper, we propose a combination of deep Q-Network and a Monte Carlo (MC) approach. The idea is to learn, using DQN, a distribution of the placement solution, on which a MC-based search technique is applied. This improves the solution space exploration, and achieves a faster convergence of the placement decision, and thus a safer learning. The obtained results show that DQN with only 8 MC iterations achieves up to 44% improvement compared with a baseline First-Fit strategy, and up to 15% compared to a MC strategy.**

***Keywords*—Deep Reinforcement Learning, Monte-Carlo, Dynamic service placement, Network slicing.**

I. Introduction

Network operators have been focusing for several years on evolving their networks, with the objective of reducing operational and investment costs, Among these evolutions, the virtualization of network functions (VNFs) [1], or more generally services, has brought the most in-depth changes to the way the network is managed leading to automation.

Supporting a wide range of services, such as Augmented Reality, Vehicle to Everything and massive Internet of Things, within the same substrate network remains a major challenge. To achieve such an objective, and further automate the network, the concept of network slicing has been introduced.

The problem of network slicing refers to the placement of constrained services [2]. This research problem generalizes the NP-hard bin-packing problem extensively studied in the literature. Indeed, service placement, as envisioned by 5G, implies not only the placement of nodes, representing the components of the service, but also links representing the network constraints existing between these different components (i.e., services in the form of a Virtual Network Function Forwarding Graph. This placement is called virtual network embedding.

Traditional solutions currently deployed in slices scheduling policies mostly follow a set of rules for node embedding. For example, Kubernetes [3] follows a two-step operation for node selection: filtering and scoring. In the filtering step, the scheduler finds the set of feasible nodes having enough available resource to meet a specific resource requests, and in the scoring step, the scheduler ranks the remaining nodes to choose the most suitable placement based on a set of active scoring rules. The drawbacks of similar heuristics is that they offer sub-optimal solutions and lack flexibility for dynamic virtual network slice requests (VNR) with stochastic arrivals/departures' processes.

Several studies in the literature have proposed VNE algorithms based on deep reinforcement learning (DRL) to counter existing shortcomings in heuristics. However, these solutions require a lot of time to converge to an optimal solution, which is not possible in real-time applications. Another component to consider is the safety of the proposed solutions when used in real-world scenarios [4].

In this paper, we try to improve the robustness and resilience, and consequently the safety of DRL placement strategies with dynamic arrival/departure of VNRs. We consider the Deep Q-Network (DQN) strategy [5] as a deep learning strategy to generate a distribution of the best placement solutions, on which a Monte Carlo strategy is applied. The combination of these two approaches allows for a more efficient exploration of the solution space. The main contributions of this paper are:

- This paper proposes a robust VNE algorithm based on DRL in a dynamic and stochastic system with VNR arrivals/departures following a Poisson process.
- We apply DRL for node embedding with the aim of meeting the computational and link bandwidth constraints of the services' components. We chose DQN as a DRL algorithm and compared it with the First-Fit strategy.
- We use MC to increase the exploration of the learning, improving thus the robustness of the DRL agent solutions. The obtained results show the advantage of combining DQN and MC in terms of achievable revenue-to-the-cost (R2C) and learning speed. This shows that our proposal improves the performance of heuristics such as First-Fit, but also improves an approach based on pure MC.
- For features extraction, we evaluate a fully connected 3 layers neural network and a Graph Convolutional Neural (GCN) network. The results found an improvement of the results with GCN at the cost of simulation time.

Next, Section II presents a literature review. Section III introduces the system model and the VNE problem. Section IV details the RL environment, the feature extraction, and the proposed VNE strategies. Section V shows the experimental setup and the results. Finally, Section VI concludes the paper.

II. Related Work

In recent years, VNE algorithms have been widely studied in the literature [6], and can be classified into three main categories: optimization-based approaches, heuristics, and machine learning-based strategies. In the following, we examine these different classes and analyze some important papers.

A. Optimization-based solutions

Given that the VNE problem is not new, a significant number of works have been proposed in this category [7]. The existing works are generally based on the formulation of the placement problem as an Integer Linear Problem (ILP), which is then solved for small-sized network instances.

Authors in [8] proposed a mixed integer regularization algorithm. In this algorithm, both nodes and link mapping are considered as a whole and modeled as a mixed integer problem. The problem is relaxed into a Linear one and divided into certainty and randomness algorithms. Vhub linear programming method is adopted in reference [9]. The VNE problem is treated as a mixed integer programming problem using the p-hub median method. The best location of VNE can be determined after the location problem of hub is solved. However, these approaches require excessive computational resources. Authors in [10] proposed an algorithm based on ILP that jointly focused on the full-resource utility and request survivability. Despite using a heuristic to accelerate the execution, the running time is still unacceptable and needs to be further reduced for real-time service placements. More recently, the authors, in [11], proposed to separate the placement problem into two parts: the resources' allocation problem and the routing problem. In order to speed up the computation of the routes, the authors proposed to use the K shortest path, which reduced significantly the computation time. While efficient and powerful, the problem separation and the restriction to the use of the K shortest paths can lead to a sub-optimal solution.

B. Heuristics

One of the approaches to reduce time cost of ILP solutions in VNE is the use of heuristic methods that offer approximate solutions with acceptable time cost. A topology-aware node ranking method is proposed in [12]. This method sorts substrate nodes and virtual nodes using the rules inspired by the Google's Pagerank algorithm [13] and then embeds virtual nodes onto substrate nodes with similar ranking positions using the "big-to-big" and "small-to-small" strategy. However, the drawback of this strategy is that node ranking is fixed per network topology, which means that the embedding decisions are hard to be optimized unless the ranking rules are changed.

Authors in [14] considered a game theory-based strategy in which the VNE problem is solved using a coordination game. Each substrate node is considered as a player that tries to achieve a Nash Equilibrium for optimal embedding solutions, while sharing the same utility function. However, the node and link embeddings are separated as individual games, which results in a lack of coordination between node and link embedding decisions, which also harms time efficiency.

A Greedy-based load balancing strategy is proposed in [15]. The main disadvantage of this solution is that the metric is scenario specific and the performance will probably drop as the environment changes. Authors in [16] maximized the resource utilization for dynamically changing requests using a global and a local fitness value functions. However, this approach used subsets of substrate in order to reduce the computational complexity, which results in a sub-optimal solution.

There also exist metaheuristic methods for a larger search space where the VNE problem is considered as a combinatorial optimization problem, such as in [17], where the authors adopted particle swarm optimization as a stochastic global optimizer. However, heuristic and metaheuristic solutions in VNE are usually designed manually according to a given scenario and are not compatible with other VNE scenarios. These drawbacks cause a reduction of both the service provider revenue, and the quality-of-experience of end users.

C. Machine Learning solutions

The above-mentioned heuristic methods to solve the VNE problem cannot fully consider the real situation of the network. Most of these solutions are based on empirical rules and cannot optimize network parameters, which mainly leads to local minimums. At present, a large number of solutions have used machine learning algorithms to solve the VNE problem. Authors in [18] introduced the application of neural networks for a dynamic allocation of physical network resources to the virtual networks. This algorithm proposes an autonomous system that improves the resource utilization by acting on nodes mapping. Authors in [19] proposed the use of Temporal-difference to learn the embedding solution that maximizes the long-term revenue. In reference [20], Q-learning algorithm is used to allocate time slot and modulation coding scheme for a data transmission with transmitted data size being the reward function. Authors in [21] propose to use the Policy Gradient algorithm to gradually learn the optimal mapping mechanism. The algorithm applies the Policy Gradient method to the VNE domain and mainly to the node embedding step. This model learns how to strike a balance between exploring better solutions and developing existing models.

There exist also approaches that combine RL and heuristics. In [22], a Monte Carlo Tree Search (MCTS) strategy is proposed. It allows to find a sub-optimal solution to the placement problem, whereas the cost of a new research remains substantial since there is no learning. In [23], the authors proposed to combine DRL with a heuristic, to make the placement safer at the cost, however, of effectiveness.

It should be noted that these strategies either consider learning on placing a set of static requests, or does not consider a large state-space (for solutions with Q-table). Moreover, the problem with reinforcement learning strategies is the safety in the decision-making process. In this paper, we consider the case of a VNE problem with dynamic arrival and departure requests with a large state space, and we offer a robust solution by improving the exploration/exploitation using new techniques that will be detailed in the following sections.

III. System Model

In this section, we describe the substrate network, the VNR and their resources. Then, we provide a mathematical description of the VNE problem and define critical RL elements.

A. General Description

Network slicing consists of building virtual network services on top of one physical network, the substrate network (Figure 1). We model the substrate as an undirected graph $\mathcal{G}_s = (\mathrm{N}_s, \mathrm{L}_s)$, where N_s and L_s denote the set of substrate nodes and links, respectively. We denote the CPU capacity of a substrate node by c_{n^s} with $n^s \in \mathrm{N}_s$, and the bandwidth capacity (BW) of a substrate link by b_{l^s} with $l^s \in \mathrm{L}_s$.

We assume that the VNRs arrive dynamically at the arrival rate λ, each with a different CPU and BW request, and a lifetime in time units, following an exponential distribution.

B. VNE Problem

The VNE problem can be defined as mapping the virtual graph $\mathcal{G}_v = (\mathrm{N}_v, \mathrm{L}_v)$ to a subgraph $\mathcal{G}_s^{'} = (\mathrm{N}_s^{'}, \mathrm{L}_s^{'})$, where N_v and L_v denote the set of VNR nodes and links, respectively. The mapping procedure can be decomposed into two stages: 1) the node mapping procedure for hosting virtual nodes from the VNR on substrate nodes with sufficient resources, and 2) the link mapping procedure for assigning virtual links (VL) onto loop-free paths of the substrate network while satisfying virtual link resource requests. It is worth noting that virtual nodes from different VNRs can share the same substrate node, and a virtual link cannot only share substrate links with other virtual links, but may also cross over multiple substrate links that form a substrate path between source and target nodes. In the case of crossing case, the substrate bandwidth of a virtual link takes up more than the resources it initially requires, which is decided by the length of the crossed substrate path, which impacts the revenue explained later in this section.

We assume that a VNR is successfully deployed if the Virtual Nodes Mapping (VNM) and the Virtual Links Mapping (VLM) to the substrate network meets its CPU and BW requirements, respectively.

$$f_{VNM} : N^v \rightarrow N^S \tag{1}$$

$$f_{VLM} : L^v \rightarrow L^S \tag{2}$$

We consider that the VNM function f_{VNM} is injective, that is, two VNFs of the same VNR can not be hosted by the same substrate network. The following equations ensure that the resource constraints are met for both CPU and BW resources:

$$c_{n^v} \leq c_{f_{VNM}(n^v)}, \forall n^v \in \mathrm{N}^v \tag{3}$$

$$b_{l^v} \leq \min_{l^v \in f_{VLM}(l^v)} b_{f_{VML}(l^v)} \tag{4}$$

where c_{n^v} denotes the CPU request of a VNR node $n^v \in \mathrm{N}_v$, and b_{l^s} the BW request of a VNR link $l^v \in \mathrm{L}_v$.

The management and orchestration of this placement is controlled by an intelligent framework compatible with the latest 5GPPP architectures [24]. We consider that the intelligent embedding function in Figure 1 is guided by the ETSI-ENI Network Function Virtualization Orchestrator (NFVO).

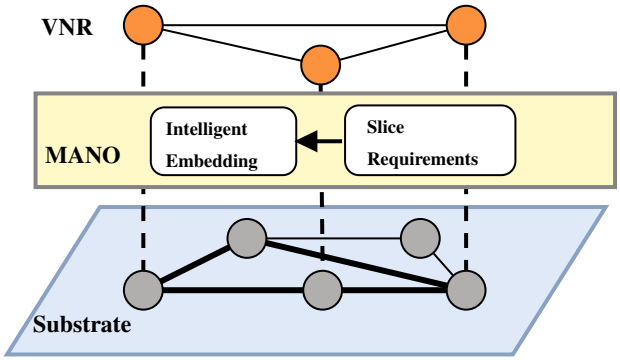

Fig. 1. System model

C. MDP model

For VNE problems, it is impossible to find a solution using supervised learning algorithms [25]. For this reason, reinforcement learning (RL) algorithms are used. RL algorithms allow to learn optimal action based on a reward function. We usually assume that there is Markov property between state probability transitions, and hence we use a Markov Decision Process (MDP) to model RL. In our VNE model, we consider that the substrate network is continuously changing and the revenue obtained by an embedding can be obtained after each decision. Hence, the node embedding problem can be modeled as an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ with a finite set of states $\mathcal{S}$ and action spaces $\mathcal{A}$, transition dynamics $\mathcal{P}$ and a reward function $\mathcal{R}$. The state transition probability can be given by:

$$P_{s_{t+1}, s_t} = P[s_{t+1} | s_t, a_t] \tag{5}$$

where s_t and a_t are the state and the action at time t. The reward $R_{s_t}^{a_t}$ at time t is obtained after selecting a_t at state s_t. The discounted reward $\hat{R}_t$ is:

$$\hat{R}_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{6}$$

Additionally, in RL algorithms the agent's task is to find the best policy π that maximizes the reward function. In MDP, there are two value functions that can be used for this aim.

1) State-value function: The state-value function $V_\pi(s)$ is only related to the current state s and is defined as the expected total reward if the agent starts its progress with the state s:

$$V_\pi(s) = E_\pi[\hat{R}_t | s_t = s] \tag{7}$$

2) Action-value function: The action-value function $q_\pi(s, a)$ is related to current action a and state s, defined by:

$$q_\pi(s, a) = E_\pi[\hat{R}_t | s_t = s, a_t = a] \tag{8}$$

Generally, the value function $V_\pi(s)$ is the sum of possible $q_\pi(s, a)$ weighted by the probability π of taking an action a in the state s, which is the definition of the policy $\pi(a|s)$.

In our problem, we consider the case of continuous state space, where the usage of value-based algorithms becomes impossible. For this reason, we adopt DRL strategies, which directly optimizes the policy of actions. More details of the used DRL algorithms are given in the next section.

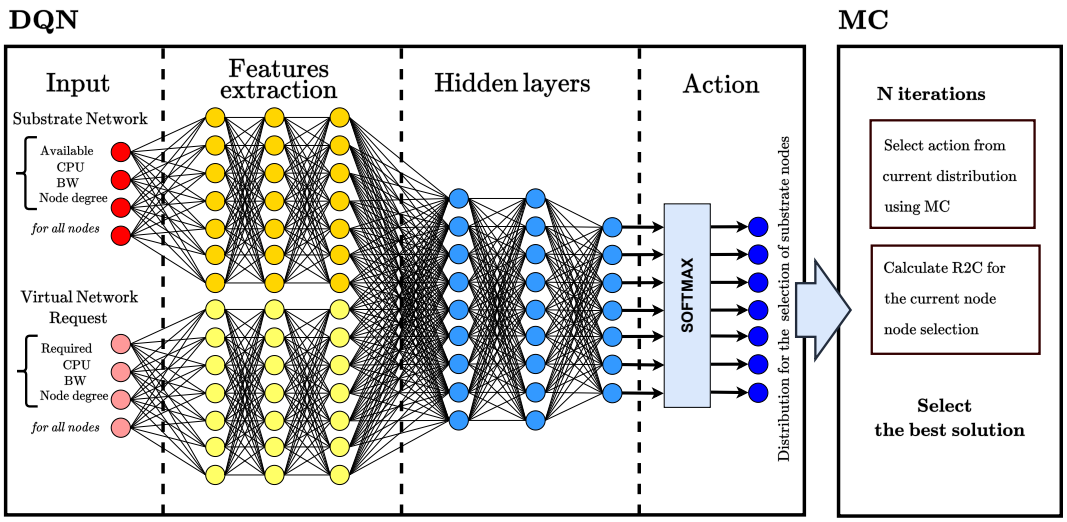

Fig. 2. Learning strategy with features extraction and Monte Carlo.

IV. A ROBUST DRL-BASED STRATEGY FOR VNE

In this section, we describe the whole process from the input to the output (Figure 2). We first describe the main learning elements in the VNE problem. Then, we present the features' extraction strategies, the embedding policies and the proposed DQN augmented with the Monte-Carlo-based strategy.

A. VNE RL Environment

We define the three main component of the RL framework for the VNE problem: the state, the action, and the reward.

1) State Representation: We consider the state representation as the raw input that will be used in the features' extraction phase. We define the state as the real-time representation of the substrate network status and the current VNR.

$$\mathcal{S} = \mathcal{C}_s \times \mathcal{B}_s \times \mathcal{D}_s \times \mathcal{C}_v \times \mathcal{B}_v \times \mathcal{D}_v$$

where $\mathcal{C}_s$ is a vector representing the available CPU at each substrate node, $\mathcal{B}_s$ the available BW at each substrate link, $\mathcal{D}_s$ the vector representing the substrate nodes' degree, which gives information about the nodes' connectivity. Similarly, $\mathcal{C}_v$, $\mathcal{B}_v$, and $\mathcal{D}_v$, are the requested number of CPU, the requested BW and the node degree at each virtual node of the VNR.

2) Action Description: The action of the VNE process is expected to be a valid placement of the VNR onto a subset of the substrate network. However, the number of possible subgraphs is exponentially proportional to the number of nodes and links, which is computationally large and counterproductive for finding an embedding solution. For this reason, we decompose the VNE problem into a sequence of virtual nodes embedding. Therefore, a placement solution is performed in several steps, where in each step a single virtual node from the VNR nodes is placed on a substrate node. Once all virtual nodes are placed on substrate nodes, the link embedding tries to find the shortest path between each couple of virtual nodes. The action output of the proposed solution is then the probability distribution for substrate nodes selection. The RL agent orders the substrate nodes from the highest probability to the lowest one, and selects the node with highest probability for the current virtual node placement.

3) Reward Definition: Contrarily to linear programming solutions or supervised machine learning where the agent has a definitive indicator of a correct action. In RL solutions, the reward function tells the agent how good was the action and the aim is to maximize the long term performance by estimating the discounted accumulative rewards. A successful placement, is considered as a good action, and the reward obtained is defined as the revenue-to-the-cost metric (R2C):

$$R2C = \frac{\sum_{n^v} c_{n^v} + \sum_{l^v} b_{l^v}}{\sum_{n^v} \sum_{n^s} c^{n^s}_{n^v} + \sum_{l^v} \sum_{l^s} b^{l^s}_{l^v}} \tag{9}$$

Given that a virtual link can be mapped to a set of substrate links, the resources required to embed a VNE may be higher than the actual requested resources. The highest R2C is 1 and corresponds to the VNR placed on exactly the requested resources. The more needed resources to ensure a VNE, the lower the R2C is. In the case of a failed VNE placement, i.e., not enough resources at the substrate nodes or no possible link mapping that meets the resource constraints, R2C is considered as 0. The proposed algorithm maximizes the R2C metric while decreasing the number of time-steps required to reach stability.

B. Features Extraction

In order to vectorize the input of the substrate network state and the VNR state, we need to extract the state information. Contrarily to traditional RL, where the policy and value functions are modeled using a table, real scenarios present large state and action spaces and thus the policy and value functions are approximated using deep neural networks (DNN). This combination of RL and DNN is called Deep Reinforcement Learning (DRL).

Similarly, in a system state with several information, it is possible to use DRL to extract features from these states instead of using the raw information of the system. This will help the DRL to use a lower number of features with useful information, and find the dependencies between different nodes of the substrate network as well as for the VNR.

Generally speaking, the more the features are extracted by the reinforcement learning agent, the more the feature matrix can represent the entire substrate network. However, it is important to avoid over-fitting during features extraction in order to avoid an increased complexity. For this reason, we propose to add two features extraction layers before the DQN layer: layers for the substrate network features extraction, and layers for the VNR features extraction (see Figure 2). We evaluate two strategies for this aim: a three layers forward neural network (FNN), and a Graph convolution network (GCN).

1) Feed Forward Neural Network: In this approach, each of the features of the substrate network and the VNR graph is extracted using three fully connected forward neural network (FNN) layers. The output of the layers is aggregated and used as input to the DRL agent which will use this information to generate an action for the controller. The advantage of this feature extraction strategy is its ability to capture the system features from the state information, while being relatively inexpensive in terms of computing resources.

2) GCN: In VNE solutions, the spatial features of a substrate network topology are critical. To manage these features more effectively while preventing our model from overfitting, an alternative approach is possible. GCN is an auto feature extraction strategy based on spectral graph theory that characterizes the spatial features of a certain graph topology [26]. The GCN apply the definition of Fourier transform to the VNE features: the system state can be decomposed into a set of functions that are orthogonal to each other. Thus, GCN is more likely to capture the dependencies among the substrate/VNR nodes and is a novel neural network that learns features by gradually aggregating information in the neighborhood [27].

We describe a layer of GCN network from a message passing perspective for each node u:

- Aggregate neighbors' representations h_v to produce an intermediate representation $\hat{h}_u$.
- Transform the aggregated representation $\hat{h}_u$ with a linear projection followed by a non-linearity: $h_u = f(W_u \hat{h}_u)$.

In this paper, we use the GraphConv function proposed by the DGL library in pyTorch.

C. Embedding strategies

1) Firstfit: The VNE problem can be considered as a bin packing problem which is solved by using the first-fit algorithm. In first-fit, each of the VNR nodes is placed on the first available substrate node with resources more than or equal to the requested resources. The controller does not search for optimizing resource utilization in the system, but just allocates the VNFs to the nearest nodes available with sufficient resources. This strategy is described in Algorithm 1.

2) Deep Q-Network: In DQN, we use a NN to approximate the Q-value function. When a new VNR arrives, a system state is constructed by encoding information about the available resources of the substrate network and the degree of the nodes, as well as the requirements of the current VNR and its node mapping, so the state represents the placement problem. The resources we consider in this paper are the CPU of the nodes and the BW of the links, but other metrics could be considered.

The VNE is performed on a node-by-node basis, which means that for each virtual node placement, a deep learning step is performed. For each virtual node, the system state is fed to a DNN in two steps: First, a feature extraction is performed on the substrate and VNR states. Then, a DNN selects an action that consists of a distribution of substrate nodes. The controller uses this distribution and selects a feasible substrate node, i.e., a substrate node with sufficient resources to implement the current virtual node, which has the highest distribution among all other feasible substrate nodes.

The steps involved in the DRL using the DQN are:

1) Past experiences are stored in a replay memory, which keeps a limited number of experiences.
2) Next action is determined by the maximum output of the Q-network.
3) The loss function is the mean squared error of the predicted Q-value $Q(s, a)$ and the target Q-value $Q^*(s, a)$. This is basically a regression problem. However, we do not know the target value as we are dealing with RL.

The Q-value update is derived from the Bellman equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \; \mathit{Loss} \tag{10}$$

where α is the learning rate. With γ being the discount factor, the loss is given by:

$$\mathit{Loss} = (r(s, a) + \gamma \max_{a_t} Q(s_{t+1}, a_t) - Q(s_t, a_t)) \tag{11}$$

Note that we consider in this algorithm that two nodes of the same VNR cannot be placed on the same substrate node, which is a fairly common assumption for VNE problems [28].

3) DQN with MC: One of the main drawbacks of DRL in stochastic systems with high-dimensional states and actions is the number of experiments needed for the system to converge. In this paper, we propose to overcome this problem by increasing the exploration in a single time step using Monte-Carlo, as presented in Algorithm 2.

The MCTS approach combines the universality of random sampling with a tree-based search strategy [29]. It showed a great ability in finding the best policy in exhaustive search problems, after its breakthrough performance in AlphaGo [30]. In the case of VNE, a search tree is iteratively generated until the predefined limit N_{iter} with numerous nodes corresponding to states and actions. N_{iter} possible embeddings are explored, using DQN strategy presented in the previous section, and the R2C is evaluated for each embedding solution. However, only the set of actions that achieves the highest R2C is stored in the replay memory. This set of actions consists of a series of DRL steps to place all virtual nodes onto the substrate

Algorithm 1: VNE using First-Fit

Result: VNR nodes mapping to the substrate network.
The virtual nodes to place N^v, the virtual nodes requirements c_{n^v}, the virtual links requirements b_{l_v}, the substrate nodes N^S, and the substrate links L^S ;
for $n^v \in N^v$ **do**
  for $n^s \in N^S$ **do**
    if $c_{n^v} < c_{n^s}$ **then**
      Action: Place n^v onto n^s ;
  end
end
if *All virtual nodes are placed* **then**
  Test virtual link mapping using Dijkstra ;
  if *Nodes and links mapping done* **then**
    Slice is placed.
  else
    Slice is dropped.
  end
else
  Slice is dropped.
end

Algorithm 2: DQN with MC algorithm

Result: VNR nodes mapping to the substrate network
N_{iter}, current state s_t, number of VNF to place K ;
while $n < N_{iter}$ **do**
  for $k \in K$ **do**
    Get current state;
    Action: Get nodes distribution as a function of the current state;
    Select feasible substrate nodes (with sufficient resources, and nodes not selected previously);
    Select substrate node with highest probability;
  end
  Test virtual node mapping;
  Test virtual links mapping;
  Evaluate R2C;
end
Select VNR placement with highest R2C ;
Place the VNR;
Update the replay memory with the chosen placement;

nodes, and must offer feasible nodes and links' mapping. The advantage of the proposed policy is to balance the exploration and exploitation problem, it allows finding the most valuable nodes in the tree that maximize the R2C of an embedding. An additional advantage is that the system learns with each MC iteration using the previously stored states and actions, while maximizing the R2C of the selected embedding. The number of iteration is chosen empirically, so that it ensures a trade-off between the optimal embedding solution and the computational complexity of the learning.

V. Simulation Results

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

A. Simulation Setup

To evaluate the performance of the proposed approach, we consider the following setup:

- The substrate network follows the `btEurope` topology with 24 substrate nodes. The capacity of the substrate nodes and links is drawn uniformly from the interval $[50, 100]$.
- To generate the virtual requests, we use the Erdős–Rényi model [31]. In this model, the generated graph is defined by the number of nodes n and the probability p of creating an edge between the nodes. With, for instance, $p = 2\ln n/n$ to generate connected graphs (more precisely, this probability value goes to one as $n \to \infty$). The requested resources (CPU and BW) of the VNRs are drawn randomly following a uniform distribution from the interval $[5, 10]$. The system operates in a dynamic manner; during each time-step, a VNR arrives to the system with a mean time between arrival MTBA $\in [1, 5, 10, 20, 40]$: once a VNR is processed, the next one arrives. We considered the arrival of 20000 VNRs in the system during the experimentation. A VNR stays in the system between 5 and 10 time-steps.
- For the model architecture, it is written in Python with the Pytorch library 2, and the DGL library 3. The neural network architecture is constructed with the following hyperparameters. We set the number of features extraction layers to 3. We use then 2 fully connected layers and finally we have one head with a fully connected layer that represents, respectively, the policy network and the value network. The learning rate is set to 5×10^{-4} and the discount factor γ is set to 0.95. To train the model, the Adam optimizer was used [32].

B. DQN with MC

We first start with comparing the DQN with MC strategy under different values of the iterations' number: $N \in [1, 4, 8]$. Figure 3 shows the R2C perceived for each value of N. We observe that the DQN without MC ($N = 1$) converges after 310k time-steps with an average R2C equal to 0.64. With $N = 4$ MC iterations, DQN converges after 80k iterations with a higher average R2C equal to 0.7, and with $N = 8$ MC iterations, the DQN converges after 40k iterations with an average R2C equal to 0.73. This shows the advantages of MC iterations on the learning speed and quality of placement, with only $N = 8$ MC iterations. Indeed, the DQN agent is able to divide the experiences required for the convergence by 10, while increasing the average R2C obtained by 15%.

C. Strategies comparison

In order to assess the importance of the placement strategy, we show in Figure 4 a comparison of DQN with First-Fit and MC strategies after DQN convergence. We observe that the

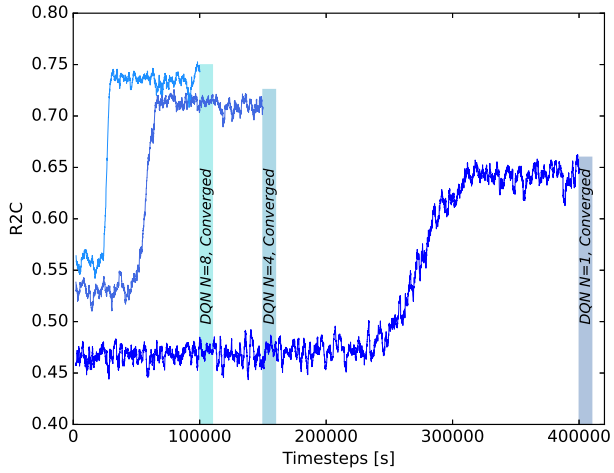

Fig. 3. Comparing the impact of MC iterations on DQN learning

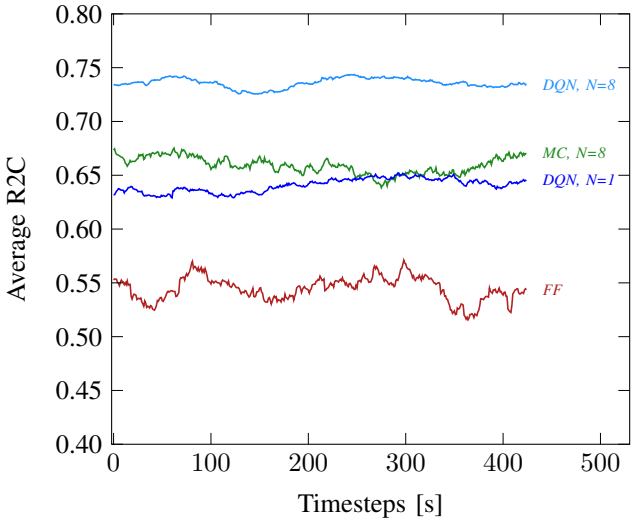

Fig. 4. Comparing FF with DQN

baseline strategy FF achieves an average R2C equal to 0.54 and MC with 8 iterations achieves an average R2C equal to 0.66. DQN with only one iteration achieves an average R2C equal to 0.64, while when learning DQN with 8 MC iterations, it achieves an average R2C equal to 0.74 which is 12% higher than MC strategy with 8 iterations, and 37% higher than the FF strategy.

Figure 5, shows the variation of the average R2C as a function of the number of MC iterations. The shaded area shows the variance of the results obtained over 5 simulations with the same parameters. We observe that when the system load is low (MTBA$\geq$ 20), DQN learns to place VNRs with an optimal R2C compared to MC. However, when the system load is very high (MTBA$=$ 1), the DQN is not able to learn the best placement solution because of the high dropping rate. The exploration action of DQN in this case increases the dropping rate compared to MC.

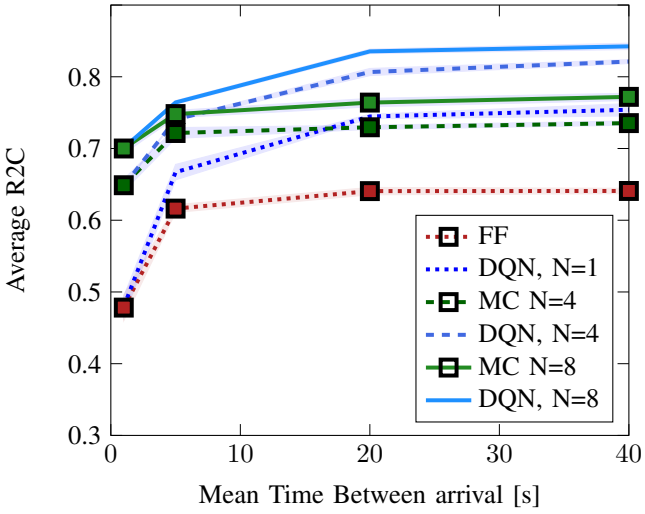

Fig. 5. Impact of the number of MC iterations on the average R2C

D. Features Extraction

In Figure 6, we compare DQN under two different feature extraction strategies: i) FNN with 3 layers and ii) two GCN Layers. The results are evaluated under different MC iterations with $N \in 1, 4, 8$. We observe that GCN achieves a higher average R2C than that obtained with FNN features extraction. GCN with 8 iterations improve significantly the system performance. Figure 7 shows the impact of MC iterations on the system performance for MTBA$=$ 20. DQN with GCN features extraction improves the average R2C by 40% when compared to FF (i.e., MC with $N = 1$), and by 15% when compared with MC with 8 iterations. However, the advantage of DQN with FNN feature extraction is to offer a higher R2C than MC with lower computational complexity.

VI. Conclusion

In this paper, we presented the problem of VNE and proposed an optimal solution using DQN with Monte-Carlo. The proposed solution, contrarily to the existing work in the literature, considers a dynamic arrival of VNRs, with a large state/action space. It improves the robustness of the solutions obtained by DRL algorithms by using Monte-Carlo for a better exploration of the possible solutions. Two feature extraction strategies were evaluated, FNN and GCN, the FNN showed a better trade-off of performance/complexity.

For our future work we will study the dynamic configuration of the number of iterations based on control theory.

References

[1] B. Yi, X. Wang, K. Li, S. k. Das, and M. Huang, "A comprehensive survey of network function virtualization," *Com. Net.*, vol. 133, pp. 212–262, 2018.

[2] P. Rost, C. Mannweiler, D. S. Michalopoulos, C. Sartori, V. Sciancalepore, N. Sastry, O. Holland, S. Tayade, B. Han, D. Bega, D. Aziz, and H. Bakker, "Network slicing to enable scalability and flexibility in 5g mobile networks," *IEEE Com. Mag.*, vol. 55, no. 5, pp. 72–79, 2017.

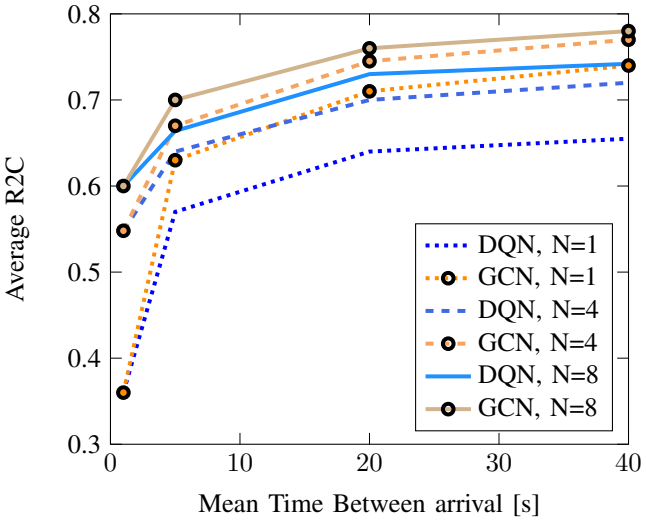

Fig. 6. Impact of features extraction on the performance of DQN

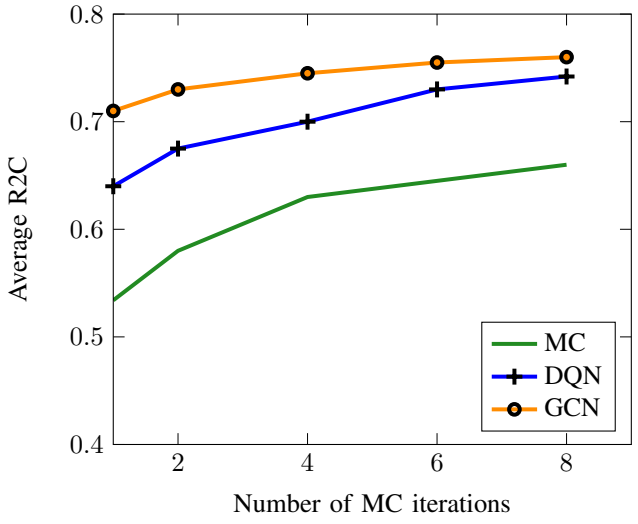

Fig. 7. Impact of number of iterations

[3] "Kubernetes scheduler," https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/, accessed: 2021-09-02.

[4] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in ai safety," *arXiv preprint arXiv:1606.06565*, 2016.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[6] S. Redana, Bulakci, C. Mannweiler, L. Gallo, A. Kousaridas, D. Navrátil, A. Tzanakaki, J. Gutiérrez, H. Karl, P. Hasselmeyer, A. Gavras, S. Parker, and E. Mutafungwa, "5G PPP Architecture Working Group - View on 5G Architecture, Version 3.0," Jun. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.3265031

[7] H. Cao, L. Yang, Z. Liu, and M. Wu, "Exact solutions of vne: A survey," *China Communications*, vol. 13, no. 6, pp. 48–62, 2016.

[8] N. M. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *IEEE INFOCOM 2009*, 2009, pp. 783–791.

[9] S. Shanbhag, A. R. Kandoor, C. Wang, R. Mettu, and T. Wolf, "Vhub: Single-stage virtual network mapping through hub location," *Computer Networks*, vol. 77, pp. 169–180, 2015.

[10] N. Shahriar, S. R. Chowdhury, R. Ahmed, A. Khan, S. Fathi, R. Boutaba, J. Mitra, and L. Liu, "Virtual network survivability through joint spare capacity allocation and embedding," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 502–518, 2018.

[11] A. Benhamiche, W. Da Silva Coelho, and N. Perrot, "Routing and Resource Assignment Problems in Future 5G Radio Access Networks," in *International Network Optimization Conference*, Avignon, France, Jun. 2019. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02401314

[12] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, and J. Wang, "Virtual network embedding through topology-aware node ranking," *ACM SIGCOMM Com. Com. Review*, vol. 41, no. 2, pp. 38–47, 2011.

[13] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[14] O. Soualah, I. Fajjari, N. Aitsaadi, and A. Mellouk, "A reliable virtual network embedding algorithm based on game theory within cloud's backbone," in *2014 IEEE International Conference on Communications (ICC)*. IEEE, 2014, pp. 2975–2981.

[15] L. Gong, Y. Wen, Z. Zhu, and T. Lee, "Toward profit-seeking virtual network embedding algorithm via global resource capacity," in *IEEE INFOCOM 2014*, 2014, pp. 1–9.

[16] C. K. Dehury and P. K. Sahoo, "Dyvine: Fitness-based dynamic virtual network embedding in cloud computing," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 5, pp. 1029–1045, 2019.

[17] Z. Zhang, X. Cheng, S. Su, Y. Wang, K. Shuang, and Y. Luo, "A unified enhanced particle swarm optimization-based virtual network embedding algorithm," *IJCS*, vol. 26, no. 8, pp. 1054–1073, 2013.

[18] R. Mijumbi, J.-L. Gorricho, J. Serrat, M. Claeys, J. Famaey, and F. De Turck, "Neural network-based autonomous allocation of resources in virtual networks," in *2014 European Conference on Networks and Communications (EuCNC)*. IEEE, 2014, pp. 1–6.

[19] S. Wang, J. Bi, J. Wu, A. V. Vasilakos, and Q. Fan, "Vne-td: A virtual network embedding algorithm based on temporal-difference learning," *Computer Networks*, vol. 161, pp. 251 – 263, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S138912861830584X

[20] Y. Kawamoto, H. Takagi, H. Nishiyama, and N. Kato, "Efficient resource allocation utilizing q-learning in multiple ua communications," *IEEE Transactions on Network Science and Engineering*, vol. 6, no. 3, pp. 293–302, 2018.

[21] H. Yao, X. Chen, M. Li, P. Zhang, and L. Wang, "A novel reinforcement learning algorithm for virtual network embedding," *Neurocomputing*, vol. 284, pp. 1–9, 2018.

[22] S. Haeri and L. Trajković, "Virtual network embedding via monte carlo tree search," *IEEE Transactions on Cybernetics*, vol. 48, no. 2, pp. 510–521, 2018.

[23] 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*, vol. 16, no. 4, pp. 1318–1331, 2019.

[24] G. P. A. W. Group, "View on 5g architecture," 5GPPP, Tech. Rep., February 2020.

[25] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[26] F. R. Chung and F. C. Graham, *Spectral graph theory*. American Mathematical Soc., 1997, no. 92.

[27] J. Liang, Y. Deng, and D. Zeng, "A deep neural network combined cnn and gcn for remote sensing scene classification," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 13, pp. 4325–4338, 2020.

[28] A. Rkhami, T. A. Quang Pham, 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, pp. 1–6.

[29] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Trans. on Computational Intelligence and AI in Games*, vol. 4, no. 1, 2012.

[30] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[31] P. Erdős and A. Rényi, "On the evolution of random graphs," *Publ. Math. Inst. Hung. Acad. Sci*, vol. 5, no. 1, pp. 17–60, 1960.

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