Evolutionary Actor-Multi-Critic Model for VNF-FG Embedding

Quang Tran Anh Pham, Yassine Hadjadj-Aoul, Abdelkader Outtagarts

Evolutionary Actor-Multi-Critic Model for VNF-FG Embedding

Pham Tran Anh Quang*, Yassine Hadjadj-Aoul*, Abdelkader Outtagarts†

*Inria, Univ Rennes, CNRS, IRISA, France
Email: quang.pham-tran-anh@inria.fr, yassine.hadjadj-aoul@irisa.fr

†Nokia-Bell Labs, France
Email: abdelkader.outtagarts@nokia-bell-labs.com

***Abstract*—The placement of Virtual Network Function – Forwarding Graphs (VNF-FGs) is one of the basic operations in the networks of the future. Being NP-hard, several heuristics and metaheuristics have been proposed. However, these approaches are inefficient due to the need to recalculate the solution at each service placement. In this paper, we adapt one of the most advanced approaches in Deep Reinforcement Learning (DRL), in order to improve exploration by generalizing the neural network calculating action values. We also propose an evolutionary algorithm to evolve these neural networks in order to discover better ones, which also avoids getting stuck in local minima. In order to avoid going through the almost innumerable number of infeasible solutions, we propose a heuristic, which combined with our DRL, makes it possible to guarantee the feasibility of the solutions and therefore to make the placement much more efficient. The simulation results we obtained confirm the quality of the solutions obtained as well as the superiority of the proposed solution over the existing one.**

I. Introduction

The advent of Network Function Virtualization (NFV) brings more flexibility than ever to current network infrastructures by providing the ability to deploy complex services on demand and in near-real time [1]. Combined with Software Defined Networking (SDN) and applied to Wide Area Networks (WAN), it is not only possible to deploy network functions at scale but also to connect them, which allows the instantiation of an entire virtual network, making the concept of network slicing into a reality [2].

Network slicing offers network Infrastructure Providers (InPs) new opportunities to get better returns on investments. Indeed, this will allow InPs to be involved in the services provisioning, and therefore to be part of the service delivery value chain, whereas they are today only a carrier network. However, network slicing raises several new challenges.

Although there are exact solutions to resolve services' placement, these approaches are still applicable only to small network instances [3]. Indeed, the problem is known to be NP-hard, even when it comes to placing simple services [4]. In this paper, we focus on the placement of more complex services, in the form of a graph, more commonly referred to as Virtual Network Function – Forwarding Graph (VNF-FG).

Several solutions have been discussed in the literature to solve the placement problem of VNF-FGs [5]. While heuristic-based approaches present a rapid convergence time [6], adding constraints or changing objectives in these approaches leads to the need of a complete redesign of the heuristics. In addition, these approaches very often get stuck in local minima. Several metaheuristics have been proposed to avoid the latter problem [7]. Although they offer a certain degree of efficiency and flexibility, these solutions have difficulty exploring the vast space of solutions and can therefore present convergence problems within limited time. Moreover, with such approaches, at each service placement a new calculation takes place, so there is no accumulation of knowledge, making the placement ineffective. Hence, our main concern in this paper is to study the potential of learning for this type of task.

Among the existing learning techniques, reinforcement learning is certainly the most appropriate approach, in view of the context encountered in networks (i.e. variable bit-rate traffic, presence of congestions, unpredictable behaviour of some services, etc.). Very few approaches in the literature have considered this type of approach for resolving the problem of VNF-FG placement. In [8], the authors proposed a distributed Q-Learning based algorithm, in which they proposed to discretize the state space, in order to avoid any explosion in the number of states. This, however, makes the approach impractical in realistic use cases.

More recently, Deep Reinforcement Learning (DRL) has proven its efficacy in tasks where the space of solutions to be explored can be very large [9], which is the case of the VNF-FG placement. One of the most efficient DRL approach, named Deep Deterministic Policy Gradient (`DDPG`) [9], considers two neural networks: the first, named Critic, is for the estimation of the value function and the second, named Actor, is for the decision making. However, even `DDPG` is not suitable for large-scale discrete action space [10].

In order to make the exploration of the solutions space more efficient, we propose in this paper a methodology to increase the number of critic networks in `DDPG`. Additionally, to avoid getting stuck in local minima, we propose not only to consider the classical learning approach for critic networks, based on the gradient descent, but also to generate a better set of critic networks using an evolutionary algorithm. Furthermore, to improve performance in the context of VNF-FG placement, we combine the proposed DRL with a heuristic, avoiding

going through the extremely large number of nonfeasible solutions, given the resource constraints of the underlying network.

The rest of this paper is structured as follows. The formulation of VNF-FGs embedding problem is presented in section II. Then, the problem is, then, reformulated as a Markov Decision Process (MDP) problem in section III. In this section, the background of the DRL and the formulation of states, actions, and reward are provided. The proposed framework is presented in section IV. The comparison of the proposed framework and other approaches is presented in section V and the conclusions of the work are presented in section VI

II. Problem Formulation

This paper focuses on embedding a set of V VNF-FGs $\mathcal{G} = \{G_1, ..., G_V\}$ into a substrate network. The sets of substrate nodes and substrate links are $\mathcal{N}$ and $\mathcal{L}$, respectively. VNF-FG ζ, G_ζ, is described in the form of a directed graph with the set of virtual network functions (VNFs) $\mathcal{N}_\zeta^{'}$ connected by the set of virtual links (VLs) $\mathcal{L}_\zeta^{'}$. VNF n' requests various types of resources denoted as a vector $\mathbf{h}_{n'} = [h_{n',0}, ..., h_{n',K_{\text{VNF}}-1}]$, where K_{VNF} is the number of types of VNF resources. In this paper, we consider three types of resources for VNFs ($K_{\text{VNF}} = 3$), i.e. the amount of central processing units (CPUs), random access memory (RAM), and storage. VL l' may request K_{VL} types of resources and QoS parameters. Three metrics ($K_{\text{VL}} = 3$) are considered, i.e. bandwidth, latency, and loss rate. The vector presented the requests of a VL is $\mathbf{f}_{l'} = [f_{l',0}, ..., f_{l',K_{\text{VL}}-1}]$. The proposed approach can be adopted to cope with more complicated scenarios, e.g. $K_{\text{VNF}} > 3, K_{\text{VL}} > 3$, by extending the states which will be discussed in next section.

A VNF is able to be deployed when its substrate host has sufficient resources

$$\sum_{n'} \Phi_n^{n'} h_{n',k} \leq r_{n,k}, \forall n, k \tag{1}$$

, where $\Phi_n^{n'} = \begin{cases} 1 & \text{, if } n' \text{ is hosted by } n \\ 0 & \text{, otherwise} \end{cases}$ and $r_{n,k}$ is the available amount of resource k at substrate node n.

The next constraints are to guarantee that each VNF is deployed at only one substrate node

$$\sum_{n} \Phi_n^{n'} \leq 1, \forall n'. \tag{2}$$

A successful VL deployment requires a successful deployment of its ends (VNFs). In addition, the requirements of the VL should be satisfied. The bandwidth related constraints are

$$\sum_{l'} \Phi_l^{l'} f_{l',bw} \leq r_{l,bw}, \forall l, \tag{3}$$

where $\Phi_l^{l'} = \begin{cases} 1 & \text{, if } l' \text{ is hosted by } l \\ 0 & \text{, otherwise} \end{cases}$ and $r_{l,bw}$ is the available amount of bandwidth at the substrate link l. Unlike the bandwidth, the accurate models for latency and loss rate are difficult to identify, especially in multi-hop environment [11]. Let us denote $D(\mathbf{\Phi}^{l'})$ and $R(\mathbf{\Phi}^{l'})$ as the actual latency and loss rate corresponding to a given mapping $\mathbf{\Phi}^{l'} = \left[\Phi_0^{l'}, \Phi_1^{l'}, ..., \Phi_{|\mathcal{L}|-1}^{l'}\right]$. Therefore, the latency and loss rate requirements $(f_{l',delay}, f_{l',loss})$ should be

$$f_{l',delay} \geq D(\mathbf{\Phi}^{l'}) \tag{4}$$

$$f_{l',loss} \geq R(\mathbf{\Phi}^{l'}) \tag{5}$$

Moreover, it requires a continuous substrate path in order to connect the substrate nodes n and m which are the host of VNFs n' and m', respectively. Following the model proposed in [12], we have the following constraints

$$\sum_{m} \Phi_{e^{nm}}^{e^{n'm'}} - \sum_{m} \Phi_{e^{mn}}^{e^{n'm'}} = \Phi_n^{n'} - \Phi_n^{m'}, \quad \forall n \in \mathcal{N}, \forall e^{n'm'} \in \mathcal{L}^{'} \tag{6}$$

We introduce an auxiliary binary variable g_ζ to indicate whether the VNF-FG ζ is deployed. The mathematical definition of g_ζ is $g_\zeta = u\left(\sum_{n' \in \mathcal{N}_\zeta^{'}} \sum_{n} \Phi_n^{n'} - |\mathcal{N}_\zeta^{'}|\right)$, where $u(x) = \begin{cases} 1 & , x \geq 0 \\ 0 & , x < 0 \end{cases}$. Consequently, $g_\zeta = 1$ when all VNFs of VNF-FG ζ are allocated. Following (6), all VLs are also allocated when all VNFs are allocated. We have

$$\sum_{n' \in \mathcal{N}_\zeta^{'}} \sum_{n} \Phi_n^{n'} \geq |\mathcal{N}_\zeta^{'}| g_\zeta. \tag{7}$$

The objective is to maximize the acceptance ratio of VNF-FGs which is $\alpha = \frac{\sum_\zeta g_\zeta}{V}$

Problem 1 (VNF-FG Allocation)**.**

$$\begin{aligned} \max \quad & \alpha \\ \textit{s.t.} \quad & (1), (2), (3), (4), (5), (6), (7) \end{aligned}$$

Due to the difficulties in determining the model of the latency and loss rate ((4) and (5)) in multi-hop network, it is not trivial to solve the above optimization problem. Without considering the QoS constraints, this optimization problem can be solved by using Integer Linear Programming (ILP) solvers or heuristic algorithms [12], [13]. Thanks to abilities of DRL in learning complex characteristics of networking [14], it is able to determine the sub-optimal VNF-FG allocation.

III. Deep Reinforcement Learning Agent

A standard reinforcement learning setup comprises a learning agent, which interacts with an environment E in discrete time-steps. The learning agent is able to enhance its performance based on its experience. At every time-step t, the learning agent observes an observation $\mathbf{o}_t$ and identifies an action $\mathbf{a}_t$ based on the observation. This action, then, is executed in E and the agent obtains the reward r_t depending on the quality of $\mathbf{a}_t$. When the environment is fully-observed, the state at time-step t, $\mathbf{s}_t$, is $\mathbf{o}_t$ and we can model the environment E as a Markov decision process (MDP) with the state space $\mathcal{S}$, the action space $\mathcal{A}$, initial state distribution

$p(\mathbf{s}_1)$, transition probability $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, and reward function $r(\mathbf{s}_t, \mathbf{a}_t)$. This assumption has been adopted to cope with VNF-FG embedding problems [15]. A policy π of the agent is to map the states to a distribution of the action $\pi : \mathcal{S} \rightarrow \mathcal{A}$. The objective of the learning agent is to determine the optimal policy π^* in order to maximize the expected reward.

In this paper, we define the states, actions, and rewards as follows.

States: in routing problems, the traffic matrix has been utilized as the states of the system [14]. Similarly, we will formulate the descriptions of VNF-FGs as the states. The VNF-FGs are described in the vector of size $|\mathcal{N}'| \times K_{\text{VNF}} + |\mathcal{L}'| \times K_{\text{VL}}$. The first $|\mathcal{N}'| \times K_{\text{VNF}}$ entries are the requests of VNFs while $|\mathcal{L}'| \times K_{\text{VL}}$ entries describe QoS requests of VLs. This vector is fed into a DRL agent in order to determine an action which expresses the mapping of VNF-FGs to the substrate networks.

Actions: we introduce auxiliary variables $a_t^{n,n'}, w_t^{l,l'} \in [0, 1]$ indicating the favor of assigning VNF n' at substrate node n and the weights of link l regrading to VL l' in time-step t, respectively. Consequently, $\mathbf{a}_t = \left[a_t^{n,n'}, w_t^{l,l'} | \forall n, n', l, l'\right]$ and the size of an action is $|\mathcal{N}| \times |\mathcal{N}'| + |\mathcal{L}| \times |\mathcal{L}'|$. To convert the action to embedding strategy (i.e. binary variables $\Phi_n^{n'}$ and $\Phi_l^{l'}$), we propose an algorithm, called Weighted First Fit Algorithm (WF^{2}A) which will be presented thoroughly in the next Section.

Rewards: A VNF-FG is deployed when all VNFs and VLs are deployed successfully. A ratio of the number of successfully deployed VNF-FGs to the number of VNF-FGs is the acceptance ratio which has been adopted to assess the performance of VNF-FG embedding algorithms in [12]. The greater acceptance ratio means more services can be deployed; therefore, we adopt the acceptance ratio as the reward.

To identify the optimal policy π^*, the most well-known DRL approach is Deep Q Networks [16]. However, it is not able to cope with problems having a large-scale action space [9]. DDPG [9] have been proposed to deal with continuous control problem; however, we will show that DDPG has a poor performance in large dimension action space in Section V. Consequently, we propose an algorithm which is able to achieve a better performance in the next section.

IV. Evolutionary Actor-Multi-Critic Model (EAMCM)

The operation of EAMCM is presented in Fig. 1. EAMCM modifies the operation of DDPG so that it is able to offer a better performance in networking problem. There are four main modifications: Enhanced exploration, WF^{2}A, MCN Evaluator, and Evolutionary Function.

A. Enhanced Exploration

The DRL agent initializes H Ornstein–Uhlenbeck (OU) processes. Then, the noisy action $\tilde{\mathbf{a}}_{t,h}$ is the combination of the proto-action with h^{th} OU process, $\tilde{\mathbf{a}}_{t,h} = \mathbf{a}_{t,p} + \mathcal{N}_h(t)$, where $\mathbf{a}_{t,p}$ is the proto action which can be determined by the actor network. Thanks to multiple noisy actions, the action space could be explored better.

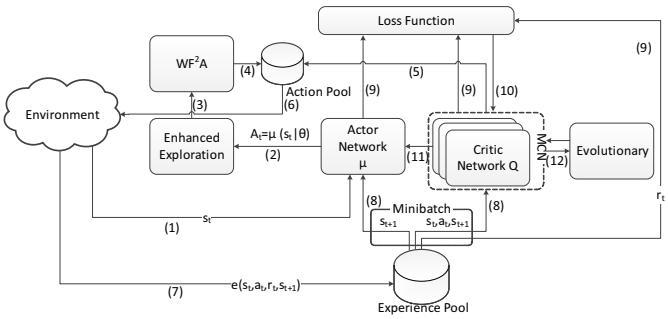

Fig. 1: EAMCM

B. Weighted First Fit Algorithm – WF^{2}A

Algorithm 1: Weighted First Fit Algorithm – WF^{2}A

1 **Input:** $\tilde{\mathbf{a}}_t$
2 **Output:** $\Phi_n^{n'}$,$\Phi_l^{l'}$
3 $\tilde{\mathbf{a}}_t^{n'}, \tilde{\mathbf{w}}_t^{l'} = \tilde{\mathbf{a}}_t$;
4 **foreach** $n_0^{'}$ **do**
5 Sort $\tilde{\mathbf{a}}_t^{n_0^{'}}$ in descending order;
6 **foreach** $\tilde{a}_t^{n_0,n_0^{'}} \in \tilde{\mathbf{a}}_t^{n_0^{'}}$ **do**
7 **if** $h_{n_0^{'},k} \leq r_{n_0,k}, \forall k$ **then**
8 $\Phi_{n_0}^{n_0^{'}} = 1$, $\Phi_n^{n_0^{'}} = 0, \forall n \neq n_0$;
9 $r_{n_0,k} = r_{n_0,k} - h_{n_0^{'},k}, \forall k$;
10 **break**;
11 **foreach** l' **do**
12 $n' \leftarrow$ Source(l'),$m' \leftarrow$ Destination(l');
13 $n \leftarrow \arg\max_n \Phi_n^{n'}$, $m \leftarrow \arg\max_n \Phi_n^{m'}$;
14 Load weights of substrate links from $\tilde{\mathbf{w}}_t^{l'}$;
15 **foreach** l **do**
16 **if** $r_{l,bw} < f_{l',bw}$ **then** $\tilde{w}_t^{l,l'} \leftarrow 1$;
17 Substrate path $P \leftarrow$ Dijkstra(n, m);
18 Based on P, set $\Phi_l^{l'}$;

In [10], the authors claimed that DDPG is not suitable for large-scale discrete action space and proposed action embedding process to convert a proto action (fractional value) to k-nearest integral solutions. However, the action embedding process costs a lot of computing resources when the dimension of actions is huge. In addition, the solutions obtained by action embedding process proposed in [10] may be unfeasible due to the constraints of substrate networks. We propose a light-weight algorithm, WF^{2}A, in order to address this problem efficiently. WF^{2}A determines a feasible allocation policy for VNFs based on the value of $\tilde{a}_t^{n,n'}$ from the output of DRL agent. The algorithm is described in Alg. 1. From line 4 to line 10, the algorithm determines the embedding of VNFs to the substrate nodes. For each VNF $n_0^{'}$, the substrate nodes are sorted in descending order of $\tilde{a}_t^{n,n_0}$. The VNF will be allocated at the substrate node if it has sufficient resources

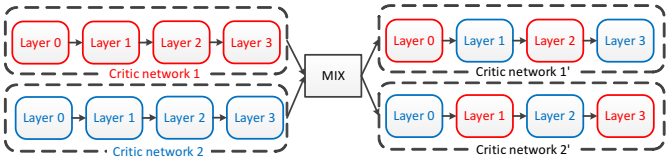

Fig. 2: MIX operator

(line 7 to 10) and the amounts of resources are updated (line 9). When all VNFs are considered, the algorithm determines the shortest path in terms of weights $\tilde{w}_t^{l,l'}$ for each virtual link l' (line 11 to line 18). The sub-process from line 15 to line 16 is to prevent the insufficient bandwidth links from the selection.

C. Evolutionary Function

To enhance the performance of the DRL agent, we propose an evolutionary function, which is able to generate a better set of critic networks. The critic networks are ranked in terms of their loss. The highest loss critic networks will be replaced by the new critic networks while the lowest loss critic networks are exploited to generate the new critic networks. Alg. 2 presents the details of this process. The function gets the set of critic networks and returns the set of evolved critic networks. First, the critic networks from the input are ranked in terms of their loss (line 3). Then, the function select $2 \times K_c$ greatest loss and lowest loss critic networks to form $\mathbb{Q}^b$ and $\mathbb{Q}^g$, respectively (line $4, 5$). The function creates K_c couples based on $\mathbb{Q}^g$ (line 6). The function creates novel critic networks by executing MIX operator and adds these critic networks to $\mathbb{Q}^n$ (line 7 to 9). MIX operator gets two critic networks as its input. It selects randomly a number of layers where the swapping occurs. These layers are then swapped between the critic networks to generate two novel critic networks inherited the parameters of the original critic networks (line 8). Fig. 2 presents an example of MIX operator. In this example, two layers are selected for swapping (layer 1 and layer 3). The evolutionary function updates critic networks in $\mathbb{Q}^b$ by the critic networks generated by MIX operator (line $10 - 11$). The parameters are updated with the coefficient η as follows $\theta^{Q_k^e} = \eta\theta^{Q_k^n} + (1-\eta)\,\theta^{Q_k^b}$. The large value of η will lead to instability of the set of critic networks. Consequently, we select the value of 0.01 for η in order to enhance the quality of critic networks gradually. The algorithm returns the evolutionary critic networks by combining $\mathbb{Q}^g$ and $\mathbb{Q}^e$ (line 12).

D. Proposed algorithm

Alg. 3 describes the proposed Evolutionary Actor-Multi-Critic Model (EAMCM). In lines 1 and 2, K critic networks, the actor network, K target critic networks, and the target actor network are initialized. Then, the replay buffer and random processes are initialized in lines 3 and 5. From the proto action, a number of noisy actions are generated by adding the noises from random processes (line 8). Then, these noisy actions are processed by WF^{2}A (line 10) and their Q-values are computed by the critic networks (line 12). The best action in terms of the average mean Q-values is selected and the system executes it and stores the transition in the replay buffer (from line 13 to line 15). Then, the loss of each critic network is identified (from line 17 to line 19). The lowest loss critic network will be selected to update the actor network (from line 20 to line 22). The critic networks will be improved by executing evolutionary function (line 23). Finally, the target networks are updated (from line 24 to line 26).

Algorithm 2: Evolutionary Function – evol ()

1 **Input:** $Q_k, k = 1, ..., K$
2 **Output:** Evolved critic networks $\mathbb{Q}^e = \{Q_k^e, k = 1, ..., K\}$
3 Rank $Q_k, k = 1, ..., K$ in terms of its loss;
4 $2 \times K_c$ greatest loss critic networks $\rightarrow \mathbb{Q}^b$;
5 $2 \times K_c$ lowest loss critic networks $\rightarrow \mathbb{Q}^g$;
6 Form K_c couples from $\mathbb{Q}^g$ randomly $\rightarrow \mathbb{Q}^c$;
7 **foreach** (Q_{k_0}, Q_{k_1}) *in* $\mathbb{Q}^c$ **do**
8 $\quad Q_{k_0}^n, Q_{k_1}^n$ = MIX(Q_{k_0}, Q_{k_1});
9 $\quad \mathbb{Q}^n \leftarrow \left(Q_{k_0}^n, Q_{k_1}^n\right)$;
10 **foreach** Q_k^b, Q_k^n *in* $\left(\mathbb{Q}^b, \mathbb{Q}^n\right)$ **do**
11 $\quad$ Update weights Q_k^b with $Q_k^n \rightarrow \mathbb{Q}^e$;
12 $\mathbb{Q}^e = \mathbb{Q}^e \bigcup \mathbb{Q}^g$;
13 **return** $\mathbb{Q}^e$;

V. Simulation Results

For the substrate network, we consider the network topology of BtEurope [17] with 24 nodes and 37 full-duplex links. The capacity of a link is a random value of 100 Mbps, 150 Mbps, 600 Mbps, and 1 Gbps. OMNeT++ [18] (v5.4.1) was utilized to simulate traffics and obtain the latency and packet loss rate.

We execute 10 runs with different seeds of random generators. Erdős-Rényi model [19] is adopted to generate the graphs of VNF-FGs. In this model, a graph G is defined by the number of nodes n and the probability of adding a possible edge to the graph p. In [19], when $p > \frac{(1+\epsilon)\log n}{n}$, the graph G(n, p) will almost certainly be connected. Consequently, a p value of $2.0 \times \frac{\log\left(|\mathcal{N}'|\right)}{|\mathcal{N}'|}$ is selected. There are from 5 to 11 VNFs in each VNF-FG. The amounts of resources of the substrate network and the requests of VNF-FGs are listed in the Tab. I. There are 200 episodes for each run. In each episode, the number of steps is 100. The reward of each episode is the mean of rewards of every step in the episode. Note that the VNF-FG requests are different in every step. The DRL agent is set up with the parameters shown in Tab. II. The number of units of fully-connected layers is 300. We adopt the Rectified Linear Unit (ReLU) activation [20] for dense layers except the output of the actor network where the sigmoid activation is adopted so as to obtain the output in range of $(0, 1)$.

Fig. 3 presents the acceptance ratio when we apply DDPG algorithm [9] in DRL agent. Although the acceptance ratio increases, it is still very low (up to 5%); thus it is impractical

Algorithm 3: EAMCM

Randomly initialize K critic networks $Q_k\left(\mathbf{s}, \mathbf{a}|\theta^{Q_k}\right), k = 1, ..., K$ and the actor $\mu\left(\mathbf{s}|\theta^{\mu}\right)$ with weights $\theta^{Q_1}, ...\theta^{Q_K}$ and θ^{μ}.
Initialize target networks $Q_k^{'}, k = 1, ..., K$ and $\mu^{'}$ with weights $\theta^{Q_k^{'}} \leftarrow \theta^{Q_k}, k = 1, ..., K$ and $\theta^{\mu^{'}} \leftarrow \theta^{\mu}$
Initialize replay buffer R
foreach *episode = 1,...,P* **do**
  Initialize H random process $\mathcal{N}_h, h = 1, ..., H$
  Receive initial observation state s_1
  foreach *t = 1,...,T* **do**
    Generate actions $\tilde{\mathbf{a}}_{t,h} = \mu\left(\mathbf{s}_t|\theta^{\mu}\right) + \mathcal{N}_h, h = 1, ..., H$
    foreach *action* $\tilde{\mathbf{a}}_{t,h}$ **do**
      $\mathbf{\Phi}_{t,h}^{n'}, \mathbf{\Phi}_{t,h}^{l'} \leftarrow \mathtt{WF}^2\mathtt{A}\left(\tilde{\mathbf{a}}_{t,h}\right)$;
      foreach Q_k **do** $q_{t,h}^k = Q_k\left(\mathbf{s}_t, \tilde{\mathbf{a}}_{t,h}|\theta^{Q_k}\right)$;
      $\mathbb{Q}_t \leftarrow \bar{q}_{t,h} = \frac{1}{K}\sum_{k=1}^{K} q_{t,h}^k$
    $h^* \leftarrow \arg\max \mathbb{Q}_t$
    $E\left(\mathbf{\Phi}_{t,h^*}^{n'}, \mathbf{\Phi}_{t,h^*}^{l'}\right) \rightarrow r_t, \mathbf{s}_{t+1}$;
    $\left(\mathbf{s}_t, \tilde{\mathbf{a}}_{t,h^*}, r_t, \mathbf{s}_{t+1}\right) \rightarrow R$
    Sample N transitions $\left(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}_{i+1}\right)$ from R
    foreach *critic network k* **do**
      $y_{i,k} = r_i + \gamma Q^{'}\left(\mathbf{s}_{i+1}, \mu^{'}\left(\mathbf{s}_{i+1}|\theta^{\mu^{'}}\right)|\theta^{Q_k^{'}}\right)$
      Update critic network k by minimizing the loss $\mathbb{L} \leftarrow L_k = \frac{1}{N}\sum_{i=1,...,N}\left(y_i - Q_k\left(\mathbf{s}_i, \mathbf{a}_i|\theta^{Q_k}\right)\right)^2$
    $k^* \leftarrow \arg\min \mathbb{L}$
    Update the actor policy using the sampled policy gradient:
    $\nabla_{\theta^{\mu}} J \approx \frac{1}{N}\sum_i \nabla_{\mathbf{a}} Q_{k^*}\left(\mathbf{s}, \mathbf{a}|\theta^{Q_{k^*}}\right)|_{\mathbf{s}=\mathbf{s}_i, \mathbf{a}=\mu(\mathbf{s}_i)} \nabla_{\theta^{\mu}}\mu\left(\mathbf{s}|\theta^{\mu}\right)|_{\mathbf{s}_i}$
    $\mathtt{evol}\left(\mathbb{Q}\right)$;
    Update the target networks:
    $\theta^{Q_k^{'}} \leftarrow \tau\theta^{Q_k} + (1-\tau)\,\theta^{Q_k^{'}}, k = 1, ..., K$
    $\theta^{\mu^{'}} \leftarrow \tau\theta^{\mu} + (1-\tau)\,\theta^{\mu^{'}}$

Parameters	Values
Substrate network	
CPU, RAM, or Storage	$0.3 - 2.0$
Link capacity	$100, 150, 600$ Mbps, and 1 Gbps
VNF-FG request	
CPU, RAM, or Storage	$0.1 - 1.0$
Bandwidth	$1 - 30$ Mbps
Latency	$1 - 100$ ms
Loss rate	$0\% - 0.5\%$

TABLE I: Parameters of Substrate networks and VNF-FG

Parameters	Values
Learning rate of actor	10^{-3}
Learning rate of critic	10^{-4}
γ	1.0
τ	0.001
Batch size	32
Optimization method	Adam [21]

TABLE II: Parameters of DRL agent

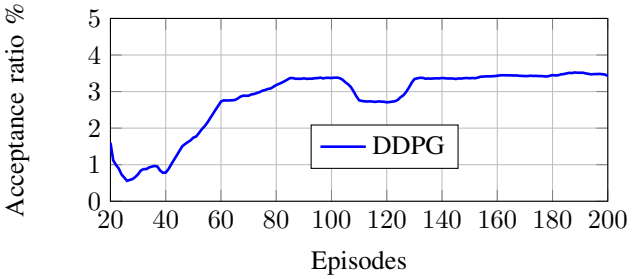

Fig. 3: Acceptance Ratio of DDPG

to deploy it in the real system. DDPG, although was designed to address continuous control environment, is not capable of coping with the huge dimensional control environment (e.g. VNF-FG embedding problem). We compare the acceptance ratio of the Actor-Multi-Critic Model (AMCM) [22], in which we do not consider the evolutionary part of EAMCM, and the ILP-based solution. The ILP-based solution is derived by solving the model in Problem 1 without considering the QoS constraints (4) and (5) since the relationship of the practical QoS metrics (e.g. latency) and traffic is not linear [23]. Moreover, it is difficult to have an accurate model for these QoS metrics in multihop networks [11]. In recent studies [13], [24], VNF-FG embedding problem was formulated as an ILP problem (without considering QoS metrics, e.g. latency and loss rate) and heuristic algorithms were proposed to determine the sub-optimal solution efficiently since computing the optimal solution for VNF-FG embedding problem costs extremely high computing resources and time. Fig. 4 presents the acceptance ratio of ILP-based solution and AMCM. Although the acceptance ratio of AMCM is lower than one of ILP at the beginning, it increases progressively and outperforms ILP after a few episodes. This is because AMCM has capabilities of learning the hidden patterns of the environment through interacting with the substrate network.

Next, we compare the performance between EAMCM and AMCM. Fig. 5 shows the performance of EAMCM and AMCM when considering the default configuration. EAMCM is slightly better than AMCM. It is because the resource constraints in default configuration is not too strict (e.g. the acceptance ratio can be up to 90%). Consequently, we conduct simulations in more difficult scenarios by reducing the capacities of links to random values of 50 Mbps, 100 Mbps, 300 Mbps or 500 Mbps. The results are shown in Fig. 6. EAMCM clearly outperforms AMCM, especially at the beginning. After 40 episodes, AMCM is able to shorten the gap which is around 5%.

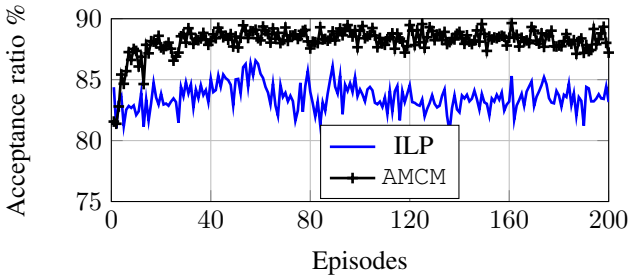

Fig. 4: Acceptance Ratio of ILP and AMCM

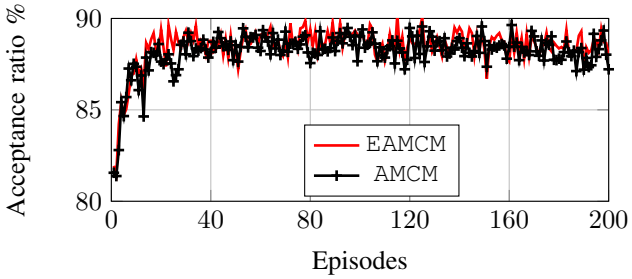

Fig. 5: Acceptance Ratio

VI. Conclusions

The issue of VNF-FGs placement is one of the recent issues that has gained the attention of the research community, with an interest not only in architectural and standardization aspects but also in the algorithmic aspects, which we address in this paper. We propose to solve this type of placement by using deep reinforcement learning, which allows an in-depth increase in knowledge unlike existing techniques which requires a recalculation from scratch for each placement. Given the very large space of possible solutions, we have proposed in this paper several techniques to improve the exploration of this space. We adapted one of the most advanced DRL approaches to improve exploration by generalizing the neural network calculating action values. We also proposed an evolutionary algorithm to evolve these neural networks in order to discover better ones, which also avoids getting stuck in local minima. To avoid going through the huge number of infeasible solutions, we proposed a heuristic guaranteeing the feasibility of the solutions and thus more efficiency.

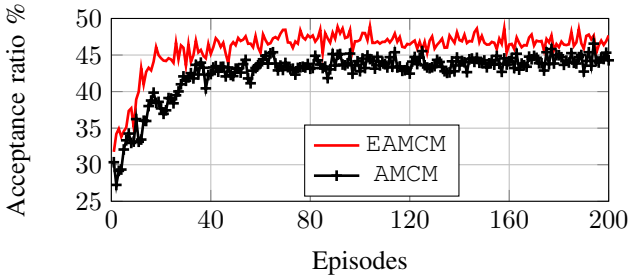

Fig. 6: Acceptance Ratio - Low Resource

References

[1] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, Feb 2015.

[2] 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*, vol. 20, no. 3, pp. 2429–2453, thirdquarter 2018.

[3] 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*, vol. 35, no. 11, pp. 2532–2541, Nov 2017.

[4] G. MIRJALILY and Z. LUO, "Optimal network function virtualization and service function chaining: A survey," *Chinese Journal of Electronics*, vol. 27, pp. 704–717, July 2018.

[5] S. Vassilaras, L. Gkatzikis, N. Liakopoulos, I. N. Stiakogiannakis, M. Qi, L. Shi, L. Liu, M. Debbah, and G. S. Paschos, "The algorithmic aspects of network slicing," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 112–119, Aug 2017.

[6] S. Khebbache, M. Hadji, and D. Zeghlache, "Scalable and cost-efficient algorithms for VNF chaining and placement problem," in *Proc. ICIN*, March 2017, pp. 92–99.

[7] ——, "A multi-objective non-dominated sorting genetic algorithm for VNF chains placement," in *Proc. IEEE CCNC*, Jan 2018, pp. 1–4.

[8] R. Mijumbi, J.-L. Gorricho, J. Serrat, M. Claeys, F. De Turck, and S. Latré, "Design and evaluation of learning algorithms for dynamic resource management in virtual networks," in *IEEE NOMS*, 2014, pp. 1–9.

[9] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015.

[10] G. Dulac-Arnold, R. Evans, P. Sunehag, and B. Coppin, "Reinforcement learning in large discrete action spaces," *CoRR*, vol. abs/1512.07679, 2015.

[11] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-driven Networking: A Deep Reinforcement Learning based Approach," in *IEEE INFOCOM*, April 2018, pp. 1871–1879.

[12] P. T. A. Quang, A. Bradai, K. D. Singh, G. Picard, and R. Riggio, "Single and Multi-Domain Adaptive Allocation Algorithms for VNF Forwarding Graph Embedding," *IEEE Transactions on Network and Service Management*, vol. 16, no. 1, pp. 98–112, March 2019.

[13] 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.

[14] G. Stampa, M. Arias, D. Sanchez-Charles, V. Muntés-Mulero, and A. Cabellos, "A deep-reinforcement learning approach for software-defined networking routing optimization," *CoRR*, vol. abs/1709.07080, 2017.

[15] Y. Xie, Z. Liu, S. Wang, and Y. Wang, "Service function chaining resource allocation: A survey," *CoRR*, vol. abs/1608.00095, 2016.

[16] 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.

[17] The internet topology zoo. [Online]. Available: http://www.topology-zoo.org/dataset.html

[18] A. Varga, "Discrete event simulation system," in *Proc. of the European Simulation Multiconference*, 2011.

[19] P. Erds and A. Rényi, "On the evolution of random graphs," *Publ. Math. Inst. Hungar. Acad. Sci*, vol. 5, pp. 17–61, 1960.

[20] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proc. AIStats*, Apr 2011, pp. 315–323.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[22] 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, December 2019.

[23] D. Tai, H. Dai, T. Zhang, and B. Liu, "On data plane latency and pseudo-TCP congestion in Software-Defined Networking," in *Proc. ACM/IEEE ANCS*, March 2016, pp. 133–134.

[24] S. Khebbache, M. Hadji, and D. Zeghlache, "Virtualized network functions chaining and routing algorithms," *Computer Networks*, vol. 114, pp. 95 – 110, 2017.