A Deep Reinforcement Learning Approach for VNF Forwarding Graph Embedding

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

A deep reinforcement learning approach for VNF Forwarding Graph 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

Network Function Virtualization (NFV) and service orchestration simplify the deployment and management of network and telecommunication services. The deployment of these services requires, typically, the allocation of Virtual Network Function - Forwarding Graph (VNF-FG), which implies not only the fulfillment of the service's requirements in terms of Quality of Service (QoS), but also considering the constraints of the underlying infrastructure. This topic has been well-studied in existing literature, however, its complexity and uncertainty of available information unveil challenges for researchers and engineers. In this paper, we explore the potential of reinforcement learning techniques for the placement of VNF-FGs. However, it turns out that even the most well-known learning technique is ineffective in the context of a large-scale action space. In this respect, we propose approaches to find out feasible solutions while improving significantly the exploration of the action space. The simulation results clearly show the effectiveness of the proposed learning approach for this category of problems. Moreover, thanks to the deep learning process, the performance of the proposed approach is improved over time.

Index Terms

Network function virtualization, VNF-FG embedding, Deep reinforcement learning, Quality of Services

I. Introduction

Dealing with new, constrained services, which must be deployed almost instantaneously, is one of the most important challenges facing Infrastructure Providers (InPs) [1]. In order to meet these new requirements, InPs have shifted from hardware solutions known to be rigid to much more agile and scalable software solutions. This vision has been fulfilled with the emergence of network function virtualization, which has made it possible to deploy new services on-demand and in near real-time [2]. However, the allocation of these services on an infrastructure remains very complex due to the services' constraints. This complexity is all the more pronounced when the services to be deployed are composite, often in the form of a service graph (i.e. VNF-FGs) [3].

The efficient and automatic placement of network services is certainly one of the most important technological building blocks to move towards a fully automated network (i.e. zero-touch network). Accordingly, researchers

from the academic and industrial spheres have stepped up their efforts to solve this crucial issue. Several studies have been carried out, some of them addressing architectural solutions, such as those developed by the European Telecommunications Standards Institute (ETSI), others proposing more algorithmic solutions, on which we focus in this paper.

Several solutions have already been proposed in the literature [4]. Some techniques are based on the formulation and resolution of optimization problems, which has the limitation of not always being applicable in a real context, given the latency of resolution or unsuitability in a real context[1]. Others propose quick heuristics but do not necessarily explore the immense space of possible actions [5]. Metaheuristics, and particularly, evolutionary algorithms, have also been proposed for the resolution of this type of problems [6]. These techniques, although effective, can present problems of convergence and therefore slowness. More recently, techniques based on deep learning have been used [7]. These techniques show some effectiveness but are simply ineffective when the space of actions is very wide. Moreover, as we show in this paper, even the most advanced learning techniques are unable to give a satisfactory answer without an adaptation to the problem addressed.

In this work, we address allocation of VNF-FGs for realizing network services. We model the VNF-FG allocation problem as a Markov Decision Process (MDP) and solve it by a deep reinforcement learning approach. We assume that operators specify their requests in terms of VNF-FGs. Such VNF-FGs include the typical elements of a network, e.g. load–balancers, firewalls, and deep packet inspection devices. This paper extends the work presented in [8] by adopting a DRL approach to deal with QoS requirements. The objective is to allocate maximum number of VNF-FGs with their QoS requirements to physical nodes in substrate networks. The contributions of this paper are threefold:

1) we formalize the VNF-FGs allocation problem as MDP with appropriate states and actions
2) we propose a heuristic algorithm to convert unfeasible solutions to feasible solutions
3) we propose a DRL framework, named Enhanced Exploration Deep Deterministic Policy Gradient, which is based on Deep Deterministic Policy Gradient (`DDPG`) in order to enhance the performance of DRL agent

Extensive simulation results, in Section VII, confirm that the proposed DRL framework offers a significantly better performance than integer linear programming (ILP) – based placement algorithm.

The rest of this paper is structured as follows. In Sec. II, a thorough discussion of related studies is provided. The formulation of VNF-FGs embedding problem is presented in Sec. III. Then, the problem is reformulated as an MDP problem in Sec. IV. In this section, the operation of `DDPG` and the forms of states and actions are also described in details. The proposed framework is presented in Sec. V. In addition, the configurations of neural networks are discussed in Sec. VI. The comparison of the proposed framework and other approaches is presented in Sec. VII. Finally, we discuss the advantages and disadvantages of the proposed framework in Sec. VIII.

[1]The resulted latency and loss, for example, are measures induced by placement and cannot be properly included in an optimization problem.

II. Related works

The conventional service placement, which in reality comes down to a problem of bin-packing or knap-sack, is one of the most studied problems in the literature [9]. This case, although very simplified compared to the use cases we are considering in this paper, is known to be NP-hard [10].

The more realistic placement of services, like the VNF-FG placement considered in this paper, is much more complex. Indeed, the services are composite since they include several sub-services, and multi-constrained (e.g. CPU, RAM, Disk, ...). To this must be added the multiple constraints on the links (e.g. bandwidth, latency, loss, ...) connecting the sub-services or VNFs.

Although there has been a lot of work on the placement of simple services (i.e., a service composed of a single NFV) [4], much less attention has been paid to the placement of VNF-FG, which is actually a fairly recent issue. We can classify them into the following four main categories.

A. Mathematical optimisation-based approaches

The first class of techniques are based on mathematical optimization.

A significant number of papers deal with the mathematical modeling of the problem in the case of one or multiple objectives. Different techniques are also used, depending on the characteristics of the problem. Indeed, some formulate the optimization problem as an Integer Linear Program (ILP) [11], others as a Mixed Integer Linear Program (MILP) [12] and others as an Integer Non-Linear Program (INLP) [13]. The problem in question being NP-hard, the resolution only works for very small network instances, so the majority of approaches stop at modeling and propose more efficient heuristics for solving the problem, so they fall into another category. Furthermore, the problem was modeled with strong assumptions. For instance, the authors in [11] assumed that each physical link has a particular delay which is not completely true. A similar assumption of loss rate can be found in [14]. In fact, latency and traffic traversing through the link have a non-linear relation [15].

In order to reduce the complexity of the resolution, different strategies are considered. After having formulated the problem of placing Service Function Chain (SFC) as a multi-objective MILP, the authors in [12], propose to reduce the objectives into a single one using the ϵ-constraint method. To solve the problem, which remains NP-hard, in polynomial time, the authors propose a linear relaxation and rounding to approximate the optimal solution of the MILP. In [16], the authors formulated the problem as an ILP, for which they proposed a solution, based on randomized Linear Program-rounding approach, resulting in a logarithmic approximation factor of the optimal. In [17], the authors formulated a VNE problem as an ILP problem with the objective to minimize the cost of provisioning bandwidth and the number of bottlenecks.

These approaches, while effective, do not guarantee that the best solutions can be found. Moreover, these approaches are subject to some limitations. Indeed, since some parameters are only obtained during run-time, these problems cannot include, for example, the latency induced by the placement or even the losses. This makes these approaches sometimes ineffective (i.e. the quality of the solutions obtained is not satisfactory) in a real context, as we show in this paper.

B. Heuristics-based approaches

Most of the papers found in the literature fall into the category of Heuristics-based solutions. The approaches belonging to this class are generally effective and could be applied in a realistic context.

In [18] and [19], the authors proposed a heuristic approach using a multi-stage graph construction, which allows representing the servers available to host the VNFs. The SFCs are then placed using a maximum flow between the vertices of the different stages of the multi-stage graph representation.

In order to avoid exploring all possible solutions, including some redundant ones (i.e. the case of symmetries), the authors, in [11], propose to guide, in an iterative process and using a binary search, the search procedure that is traditionally performed by solvers. Similarly, in [20], M. M. Tajiki et al. proposed a simple but effective heuristic consisting in iteratively placing the VNFs in series using the nearest search procedure.

Instead of considering the placement of VNFs and the links connecting them at the same time, some approaches adopt a two-step approach, in which VNFs are placed and then connected, which is a routing problem. In [21], the authors proposed the SFC-MAP algorithm, which places VNFs and connects them using the shortest path algorithm in a multi-layer graph resulted from the transformation of the network topology based on the SFC constraints. The authors, in [5], extended the eigen decomposition algorithm to deal with VNF-FG placement. The main benefit of this algorithm lies in the fact that its complexity depends mainly on the size of the physical infrastructure, making it very attractive.

Heuristics are a good alternative in systems where the context is not very changing. In systems where constraints and objectives are changing, these types of approaches are not very suitable since they generally require a total redesign of the heuristic. Moreover, with heuristics, we have a rapid convergence at the price of the risk of sticking at a local minimum that may not be effective.

C. Metaheuristics-based approaches

Several papers in the literature have considered approaches based on metaheuristics.

Most of the existing work falling in this category consider using evolutionary algorithms for the VNF-FG placement [6][22][23][24]. In [6], the authors proposed using the Non-dominated Sorting Genetic Algorithm (NSGA-II), which is known to be efficient to deal with multi-objective optimization problems.

To avoid exploring all possible combinations, which would make convergence extremely slow, the authors preferred to use modified versions in order to explore only feasible solutions. This allows substantially improved performance at the price of a loss of the genericity of the NSGA-II algorithm. While the authors, in [22] and [24], consider modifications that leave the choice of the VNFs mapping random, ensuring a good exploration of the space of solutions, other approaches have chosen to steer the mapping to accelerate the convergence. In fact, in [23], the authors proposed a greedy version of the non-dominated sorting genetic algorithm (NSGA-II), in which nodes with higher residual resources have a higher priority to be mapped; while for link mapping, shorter paths are prioritized.

Other meta-heuristics have also been used in the literature. In [25], the authors proposed a novel solution combining mathematical programming and a Variable Neighborhood Search (VNS) meta-heuristic in order to

improve the exploration of the space of solutions. With the exception of this last metaheuristic, to the best of our knowledge, there are no other papers that make use of other types of meta-heuristics for VNF-FG placement.

Metaheuristics make it possible to respond effectively to the problem VNF-FG placement. They can very easily integrate new objectives or constraints without reconsidering the solution, unlike heuristics. It should also be noted that there is some research work that shows the convergence of these algorithms towards the optimum under certain conditions [26].

D. Reinforcement learning-based approaches

Reinforcement learning (RL) is the area of learning where an agent learns to make decisions through rewards or penalties received as a result by executing one or more actions. Learning from past experiences is a valuable ability to adapt to variations in the environment (i.e. variation in traffic type, network configuration, etc.).

Recent achievements have shown the potential of these approaches to solve combinatorial optimization problems [27]. This type of approach has even demonstrated empirically to be more effective than supervised learning, even in the presence of optimal labels [28].

Several technological building blocks related to the placement of VNF-FG have been proposed in the literature. The majority have been interested in the issue of routing [29], which is a sub-problem of VNF-FG placement. However, very few studies have tackled the whole issue. In [7], the authors consider a Markov Decision Process (MDP) to model a decision making strategy consisting in placing the VNFs and, then, connecting them using the shortest path algorithm. As finding the optimal policy using such a strategy is intractable given the state space, the authors proposed a Monte Carlo Tree Search (MCTS) for solving the MDP. While efficient this approach presents a high complexity as the placement is performed sequentially for each request. In [30], the authors proposed a distributed Q-Learning based algorithm for VNF-FG allocation. In order to avoid any explosion in the number of states that would risk undermining the proposed approach, the authors proposed to discretize the state space, which makes the approach impractical in realistic use cases.

The solution we propose, in this paper, for the VNF-FG placement makes it possible to overcome the limits of scalability of the approaches introduced below, by using a much more efficient approximation thanks to the use of neural networks. In opposition with [7], the proposed solution allows placing one or multiple VNF-FG at the same time and after the learning phase, the response to a request is almost immediate.

Our approach is based on the most recent advances in the field [31] and which allow us to have better efficiency, thanks in particular to the use of the concept of “memory replay” which allows us to have the advantages of supervised learning. Moreover, in order to have a better efficiency we propose several improvements to the existing techniques which allow to better explore the space of solutions.

III. VNF-FG EMBEDDING PROBLEM

A. Problem Formulation

In this paper, a discrete time-step system is considered. At the beginning of each time-step, a central orchestrator has to deploy a number of VNF-FGs which may comprise VNFs connected by virtual links (VLs). The life-time

Notation	Description
$\mathcal{N}$	Set of substrate nodes
$\mathcal{L}$	Set of substrate links
$r_{n,k}$	Available of k^{th} resource at substrate node $n \in \mathcal{N}$
$r_{l,bw}$	Bandwidth of link $l \in \mathcal{L}$
$\mathcal{N}^{'}$	Set of VNFs of all VNF-FGs
$\mathcal{L}^{'}$	Set of VLs of all VNF-FGs
$\mathcal{N}_{\zeta}^{'}$	Set of VNFs of VNF-FG ζ
$\mathcal{L}_{\zeta}^{'}$	Set of VLs of VNF-FG ζ
$h_{n',k}$	Request of resource k of VNF n'
$h_{l',k}$	Request of resource (or QoS) k of VL l'

TABLE I: Substrate network and VNF-FG notations

of a VNF-FG could span over multiple sequential time-steps and its allocation in time-slots may be different; therefore requiring the migration of VNFs. In the papers which focus on the optimization of resource allocation, the migration of VNFs has been neglected [32]–[34]. Therefore, we assume the migration process is ideal. It means the VNF can be migrated without interrupting the running service. Each VNF requires a specific amount of resources corresponding to the number of supported users and the type of services. They could be central processing units (CPUs), Random Access Memory (RAM), Storage, Radio, etc. Let us denote K_{VNF} as the number of resource types indexed by $0, 1, \ldots, K_{\text{VNF}-1}$ and $h_{n',k}$ as the amount of resource k requested by VNF n'. The vector of requested resources of VNF n' is $\mathbf{h}_{n'} = [h_{n',0}, ..., h_{n',K_{\text{VNF}}-1}]$. The core network scenarios are considered in this paper. In these scenarios, computing tasks are major; therefore, the requests of VNFs are computing resources (i.e. CPU, RAM, and storage) ($K_{\text{VNF}} = 3$). Note that the proposed mechanism can be adopted to deal with more complicated VNF resources by extending the input of the DRL agent.

Each VL is defined by K_{VL} network oriented metrics (e.g. bandwidth, latency, loss rate, etc.). The metric k of VL l' is denoted as $h_{l',k}$ and $\mathbf{h}_{l'} = [h_{l',0}, \ldots, h_{l',K_{\text{VL}}-1}]$ as the vector of request resources of VL l'. In this paper, bandwidth, latency, and packet loss rate are the metrics of VLs ($K_{\text{VL}} = 3$).

A VNF-FG ζ can be described as a directed graph $G_{\zeta}^{'}$ with the set of VNFs and VLs denoted as $\mathcal{N'}_{\zeta}$ and $\mathcal{L'}_{\zeta}$, then the number of VNFs and the number of VLs are $|\mathcal{N}_{\zeta}^{'}|$ and $|\mathcal{L}_{\zeta}^{'}|$, respectively. The set of all VNFs and all VLs from all VNF-FGs are denoted as $\mathcal{N}'$ and $\mathcal{L}'$. The substrate network can be described as a directed graph G. The sets of nodes and directed links of the substrate network are $\mathcal{N}$ and $\mathcal{L}$, therefore the number of substrate nodes and substrate links are $|\mathcal{N}|$ and $|\mathcal{L}|$, respectively. The notations of VNF-FGs and the substrate network are provided in Table I. A VNF is successfully deployed when its substrate node has sufficient resources, i.e.

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

where $\Phi_n^{n'}$ is a binary variable indicating if n' is deployed at substrate node n and $r_{n,k}$ is the available amount of

resource k at substrate node n. Each VNF can be deployed at only one substrate node n; therefore

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

In existing literature [17], [35], only bandwidth has been taken into account in the problem formulation. In this paper, we also consider other QoS requirements since they also impact the success of VNF-FG embedding. We adopt unsplittable multi-commodity flow to model the substrate paths of VLs. A VL is successfully deployed when its VNFs are deployed and its QoS requirements are met. For bandwidth requirements, we have

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

where $\Phi_l^{l'}$ is a binary variable which is 1 if l' is deployed at the substrate link l, $r_{l,bw}$ is the available amount of bandwidth at the substrate link l. Unlike the bandwidth, it is non-trivial to derive accurate models for latency and loss rate [36], especially in multi-hop environment [37]. Consequently, the QoS constraints of latency and loss rate are difficult to be formulated. We denote $D(\mathbf{\Phi}^{l'})$ and $R(\mathbf{\Phi}^{l'})$ as the actual latency and loss rate corresponding to a given mapping $\mathbf{\Phi}^{l'} = \left[\Phi_0^{l'}, \Phi_1^{l'}, ..., \Phi_{|\mathcal{L}|-1}^{l'}\right]$. Note that the mapping $\mathbf{\Phi}^{l'}$ is to create a path to connect the head VNF and the end VNF of VL l'. This path has to satisfy the latency and loss rate requirements $(h_{l',delay}, h_{l',loss})$. Thus, we have

$$h_{l',delay} \geq D(\mathbf{\Phi}^{l'}), \forall l' \in \mathcal{L}^{'} \tag{4}$$

$$h_{l',loss} \geq R(\mathbf{\Phi}^{l'}), \forall l' \in \mathcal{L}^{'} \tag{5}$$

We denote $I(n)$ and $O(n)$ as the sets of incoming links and outgoing links of node n. The following constraints enforce a continuous substrate path connecting VNFs n', m' of virtual link l':

$$\sum_{l_a \in O(n)} \Phi_{l_a}^{l'} - \sum_{l_b \in I(n)} \Phi_{l_b}^{l'} = \Phi_n^{n'} - \Phi_n^{m'}, \quad \forall n \in \mathcal{N}, \forall l' \in \mathcal{L}^{'} \tag{6}$$

When the head VNF and the end VNF of a VL are deployed at the same substrate node, Eq. 6 can form a loop. Indeed, the right-hand side (R.H.S) will be zero since $\Phi_n^{n'} = \Phi_n^{m'} = 1$. Consequently, Eq. 6 becomes $\sum_{l_a \in O(n)} \Phi_{l_a}^{l'} - \sum_{l_b \in I(n)} \Phi_{l_b}^{l'} = 0$. It means $\sum_{l_a \in O(n)} \Phi_{l_a}^{l'} = \sum_{l_b \in I(n)} \Phi_{l_b}^{l'}$. We have two cases: (i) $\sum_{l_a \in O(n)} \Phi_{l_a}^{l'} = \sum_{l_b \in I(n)} \Phi_{l_b}^{l'} = 0$ or (ii) $\sum_{l_a \in O(n)} \Phi_{l_a}^{l'} = \sum_{l_b \in I(n)} \Phi_{l_b}^{l'} = 1$. When $\sum_{l_a \in O(n)} \Phi_{l_a}^{l'} = \sum_{l_b \in I(n)} \Phi_{l_b}^{l'} = 1$, it forms a loop. We use the constraints introduced in [20] to avoid the loop

$$\sum_{l_a \in O(n)} \Phi_{l_a}^{l'} + \sum_{l_b \in I(n)} \Phi_{l_b}^{l'} \leq 1, \forall n \in \mathcal{N}, \forall l' \in \mathcal{L}^{'} \tag{7}$$

Thanks to these constraints, the case $\sum_{l_a \in O(n)} \Phi_{l_a}^{l'} = \sum_{l_b \in I(n)} \Phi_{l_b}^{l'} = 1$ cannot occur; therefore the loop does not exist.

Let us denote $g_\zeta = u\left(\sum_{n' \in \mathcal{N}_\zeta^{'}} \sum_{n} \Phi_n^{n'} - |\mathcal{N}_\zeta^{'}|\right)$, where $u(x)$ is a step function ($u(x) = 1$ when $x \geq 0$ and $u(x) = 0$ when $x < 0$), and $\mathcal{N}_\zeta^{'}$ the set of VNFs of the VNF-FG ζ. Note that $g_\zeta = 1$ when all VNFs of VNF-FG ζ are allocated. Thanks to (6), all VLs are also allocated when all VNFs are allocated. We have:

$$\sum_{n' \in \mathcal{N}_\zeta^{'}} \sum_{n} \Phi_n^{n'} \geq |\mathcal{N}_\zeta^{'}| g_\zeta. \tag{8}$$

The objective is to maximize the number of accepted VNF-FGs. A VNF-FG is accepted if and only if all VNFs and VLs are allocated and the QoS requirements are satisfied.

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

$$\begin{aligned} \max \quad & \sum_{\zeta} g_\zeta \\ \textit{s.t.} \quad & (1), (2), (3), (4), (5), (6), (7), (8) \end{aligned}$$

Solving the above optimization problem is not trivial due to the difficulties in estimations of the real latency and loss rate in (4) and (5). Without considering the QoS constraints, this optimization problem can be solved by using ILP solvers or heuristic algorithms [8], [35]. However, it is non-trivial to model QoS metrics such as loss rate and latency. For instance, the latency of a link depends on the traffic traversing it and the relation between traffic and latency is not linear [15]. Therefore, we cannot model the problem and solve it as an ILP. Deep reinforcement learning has been proved its performance in solving networking related problems. In VNF-FG embedding problem, DRL is able to learn the non-linear relation between QoS metrics and traffic; therefore, in this paper, we exploit DRL so as to determine the sub-optimal VNF-FG allocation.

IV. Deep Reinforcement Learning agent

A. Reinforcement learning background

We consider a standard reinforcement learning setup where a learning agent interacts with an environment E in discrete time-steps and improves its performance based on its own experience. At time-step t, the agent observes an observation $\mathbf{o}_t$, computes and executes an action $\mathbf{a}_t$, and receives a reward r_t. We assume that the environment is fully-observed, thus the state at time-step t, $\mathbf{s}_t$, is $\mathbf{o}_t$. Consequently, the environment E can be modeled as a 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 $r(\mathbf{s}_t, \mathbf{a}_t)$. Note that this assumption has been widely adopted to cope with VNF-FG embedding problems [38]. The detailed descriptions of s_t and a_t depend on the problem. We will discuss further s_t and a_t in the context of VNF-FG embedding problem in Section IV-C and Section IV-D, respectively. The behavior of the agent is defined by a policy π which maps the states to a distribution of the action $\pi : \mathcal{S} \rightarrow \mathcal{A}$. The discounted reward at time-step t is defined as $R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r(\mathbf{s}_i, \mathbf{a}_i)$, where γ is a discounting factor and T is the number of steps to the terminal state or the maximum number of steps. The objective in reinforcement learning is to maximize the expected return $J = \mathbb{E}_{r_i, \mathbf{s}_i \sim E, \mathbf{a}_i \sim \pi}[R_1]$. The notations are presented in Table II.

The expected return if taking an action $\mathbf{a}_t$ in state $\mathbf{s}_t$ and following policy π is called the action-value function

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_{r_{i \geq t}, \mathbf{s}_{i>t} \sim E, \mathbf{a}_{i>t} \sim \pi}[R_t | \mathbf{s}_t, \mathbf{a}_t] \tag{9}$$

Notation	Description
$\mathcal{S}$	State space
$\mathcal{A}$	Action space
$\mathbf{s}_t$	State at time-step t
$\mathbf{a}_t$	Action at time-step t
$\pi(\mathbf{s}_t)$	Policy of DRL agent at state s_t
r_t	Reward at time-step t
$p(\mathbf{s}_1)$	Initial state distribution
$p(\mathbf{s}_{t+1}\|\mathbf{s}_t, \mathbf{a}_t)$	Transition probability
$r(\mathbf{s}_t, \mathbf{a}_t)$	Reward function
R_t	Discounted reward at time-step t
γ	Discounting factor
$Q^\pi(\mathbf{s}_t, \mathbf{a}_t)$	Action-value function

TABLE II: DRL notations

Eq. (9) can be expressed in the recursive form known as the Bellman equation as follows

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_{r_t, \mathbf{s}_{t+1} \sim E}\left[r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\mathbf{a}_{t+1} \sim \pi}\left[Q^\pi(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})\right]\right] \quad (10)$$

In an MDP, the objective is to determine an optimal policy $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$. Let us denote $Q^*(\mathbf{s}_t, \mathbf{a}_t)$ as the optimal Q-function of state-action pairs; then the optimal policy can be determined by $\pi^*(\mathbf{s}_t) = \arg\max_a Q^*(\mathbf{s}_t, \mathbf{a}_t)$. The Q-learning algorithm [39] can be utilized to find optimal values of $Q^*(\mathbf{s}_t, \mathbf{a}_t)$ through iterative processes.

The Q-learning algorithm is able to achieve an optimal policy when the state space and action space are limited. Nevertheless, it does not work when the spaces are large due to the complexity in approximating the value of $Q^*(\mathbf{s}_t, \mathbf{a}_t)$. Deep Q-learning algorithm (DQL) was proposed to overcome these challenges. In DQL, deep neural networks (DNNs) are implemented to approximate the value of $Q^*(\mathbf{s}_t, \mathbf{a}_t)$. In reinforcement learning algorithms, the average rewards could be unstable or diverge when a nonlinear approximator is applied [40]. To overcome these shortcomings, the experience replay and target Q-network has been introduced [41]. The experience replay stores transitions $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$ in a replay memory and then selects randomly a set of transitions to train DNNs. It enables that the DNN is trained with both past and recent experiences. Moreover, the correlations between transitions could be eliminated [42]. The target Q-network is similar to the Q-network except that its weights are updated after every τ steps from the Q-network and unchanged in other steps. By using the target Q-network, the stability of the training process is improved.

If the target policy is deterministic, we can rewrite Eq. (10) as follows

$$Q^\mu(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_{r_t, \mathbf{s}_{t+1} \sim E}\left[r(\mathbf{s}_t, \mathbf{a}_t) + \gamma Q^\mu(\mathbf{s}_{t+1}, \mu(\mathbf{s}_{t+1}))\right] \quad (11)$$

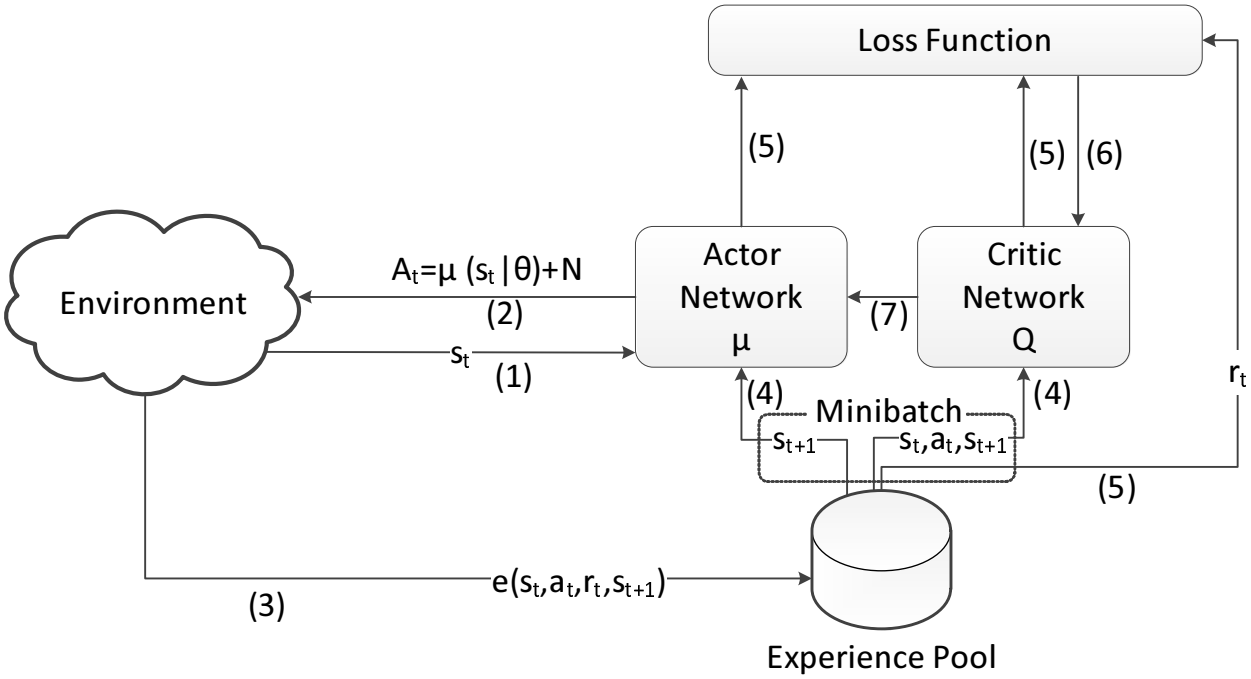

Fig. 1: DDPG

Eq. (11) depends solely on the environment; therefore, it is possible to learn Q^{μ} off-policy based on the transitions generated from a policy β. We parameterize the Q function by θ^{Q} which can be derived by minimizing the loss

$$L\left(\theta^{Q}\right)=\mathbb{E}_{\mathbf{s}_t\sim\rho^{\beta},\mathbf{a}_t\sim\beta,r_t\sim E}\left[\left(Q\left(\mathbf{s}_t,\mathbf{a}_t|\theta^{Q}\right)-y_t\right)^2\right], \tag{12}$$

where $y_t=r\left(\mathbf{s}_t,\mathbf{a}_t\right)+\gamma Q\left(\mathbf{s}_{t+1},\mu\left(\mathbf{s}_{t+1}\right)|\theta^{Q}\right)$ and ρ^{β} is the discounted state visitation distribution for policy β.

B. DDPG overview

The most well-known DRL approach is Deep Q Networks [41]. However, it is not able to cope with problems having a large-scale action space [43]. The authors in [43] proposed Deep Deterministic Policy Gradient algorithm (DDPG) to deal with large-scale action space problems. Fig. 1 describes DDPG algorithm.

In DDPG, two neural networks are maintained to learn the policy and the Q-value separately. They are the actor network (for policy) and critic network (for Q-value). There are 7 steps in each learning step:

1) The actor network acquires the current state of the environment $\mathbf{s}_t$.
2) The actor network computes a proto action $\mu\left(\mathbf{s}_t|\theta^{\mu}\right)$ with its current weights θ^{μ}. It also adds noise into the proto action and executes action $\mathbf{a}_t$.
3) After executing $\mathbf{a}_t$, the environment returns a reward r_t and the next state $\mathbf{s}_{t+1}$. The transition comprises $\mathbf{s}_t$, $\mathbf{a}_t$, r_t, and $\mathbf{s}_{t+1}$ which will be collected and stored in the experience pool.
4) A minibatch consists of a number of transitions. The actor network will retrieve the batch of $\mathbf{s}_{t+1}$ to determine the predicted actions by using its target network. The predicted actions will be utilized to compute the loss

in step 5. Meanwhile, the critic network uses the batches of $\mathbf{s}_{t+1}$ and the predicted actions from the actor network to compute the target Q-values by its target networks.

5) The loss function is computed by the temporal difference.
6) The critic network is trained with the loss value.
7) The critic Network computes the gradients and updates them to the actor network.

C. States

In routing problems, the traffic matrix has been utilized as the states of the system [44]. The traffic matrix can be seen as the requests of clients to the physical networks. We adopt a similar idea to the VNF-FG embedding problem. In VNF-FG embedding problem, the client requests are described in the form of VNF-FGs; therefore, we will formulate the descriptions of VNF-FGs as the states. A VNF-FG can be expressed as the chain of VNFs where each VNF has specific resource requirements. Besides VNFs, VLs, which connect VNFs, have specific QoS requirements, e.g. latency.

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 express the requests of computing resources of VNFs while $|\mathcal{L}'| \times K_{\text{VL}}$ entries expresses the 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.

D. Actions

We introduce auxiliary variables $a_t^{n,n'} \in [0,1]$ indicating the priority of assigning VNF n' at substrate node n in time-step t. In addition, we introduce the auxiliary variables $w_t^{l,l'}$ as the weights of links which will be exploited by Dijkstra algorithm to find the path for VL l' in time-step t. For instance, Dijkstra algorithm takes into account all weights $w_t^{l,l_0^{'}}, \forall l \in \mathcal{L}$ to identify the substrate path for virtual link l_0'. Consequently, $\mathbf{a}_t = \left[a_t^{n,n'}, w_t^{l,l'} | \forall n, n', l, l'\right]$ and the size of the action is $|\mathcal{N}| \times |\mathcal{N}^{'}| + |\mathcal{L}| \times |\mathcal{L}^{'}|$. Since it needs binary values to embed the VNF-FG to the substrate networks (i.e. $\Phi_n^{n'}$ and $\Phi_l^{l'}$), we propose an algorithm, called Heuristic Fitting Algorithm (`HFA`), to determine the embedding strategy from $a_n^{n'}$ and $w_l^{l'}$. The algorithm will be presented thoroughly in the next Section.

E. Rewards

We adopt the acceptance ratio as the reward of an action. A VNF-FG is deployed when all VNFs and VLs are deployed successfully. A VNF is deployed successfully when its resource requirements are met by the substrate host. A VL is deployed successfully when its substrate path connects the substrate hosts of its VNFs and satisfies QoS requirements (e.g. bandwidth, latency, loss rate, etc.). The requests of VNFs and VLs at time-step t are described in s_t and allocation of VNFs and VLs in the substrate network is described in a_t. The acceptance ratio (AR) has been adopted to assess the performance of VNF-FG embedding algorithms [35], [45]. The reward at time-step t (r_t) depends on the requests (s_t) and allocation (a_t). Due to the complexity of the relation between QoS metrics and traffic in multi-hop environment, it is non-trivial to retrieve the explicit reward function. However, we can measure QoS metrics; therefore, it is possible to determine the reward at each time-step.

V. Enhanced Exploration Deep Deterministic Policy Gradients

A. DDPG-HFA

DDPG [43] is not suitable for large-scale discrete action space [31]. In [31], the author proposed action embedding process to convert a proto action (fractional value) to k-nearest integral solutions; then selecting the best action by the critic network. However, the proposed approach in [31] cannot be adopted to solve the high-dimensional action space such as VNF-FG embedding problem. Moreover, in VNF-FG embedding problem, there are constraints of resources such that some discrete actions are not feasible. The solutions obtained by action embedding process proposed in [31] may be unfeasible due to the constraints of substrate networks. We propose a light-weight algorithm, HFA, in order to address this problem efficiently.

HFA determines a feasible allocation policy for VNFs based on the value of the proto action $\tilde{\mathbf{a}}_t$ from the output of the DRL agent. $\tilde{\mathbf{a}}_t$ comprises $\tilde{a}_t^{n,n'}$ and $\tilde{w}_t^{l,l'}$. The proto action for allocating VNF n' is a vector $\tilde{\mathbf{a}}_t^{n'}$. The weights of substrate links corresponding to VL l' is a vector $\tilde{\mathbf{w}}_t^{l'}$. The algorithm is described in Alg. 1. First, the algorithm extracts $\tilde{a}_t^{n'}$ and $\tilde{w}_t^{l'}$ from the action determined by the DRL agent. From line 4 to line 12, the algorithm determines the embedding of VNFs to the substrate nodes. For each VNF n', the substrate nodes are ordered in terms of the weights $\tilde{a}_t^{n,n'}$. The substrate nodes with greater weights are considered first. If the substrate node has sufficient resources, the VNF will be allocated at the substrate node (line 7 to 12). Then, the remaining resources of the substrate node are updated (line 11). After 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 13 to line 23). The sub-process from line 19 to line 21 is to prevent the insufficient bandwidth links from the selection.

The position of HFA in the proposed architecture is presented in Fig. 2. At step 2 of DDPG, the proto action is processed by HFA in order to define a solution which meets all requirements of VNFs. The later steps are similar to ones of DDPG.

B. Enhanced Exploration Deep Deterministic Policy Gradient (E^2D^2PG)

The operation of $\mathrm{E}^2\mathrm{D}^2$PG is presented in Fig. 3. In $\mathrm{E}^2\mathrm{D}^2$PG, the proto action will be processed by an Enhanced Exploration module (step 2) and a HFA module (step 3) before evaluating by an evaluator (step 5) to select the action which will be executed (step 6). In the enhanced exploration, the noise is added to the proto action to create H noisy actions. These H noisy actions and the proto action are fed into the HFA in order to determine the feasible allocation of VNFs. The actions proposed by the HFA is evaluated by the evaluator which could be a set of multiple critic networks. The best action identified by the evaluator will be executed, then the corresponding next state and reward are collected to update the critic and actor as described in DDPG. The details of the proposed modules are described as follows.

1) Enhanced exploration: The conventional DDPG [43] generates only one action per iteration; thus, it slowly explores the action space especially in large action space. To enhance exploration, we propose the enhanced exploration, which is described below.

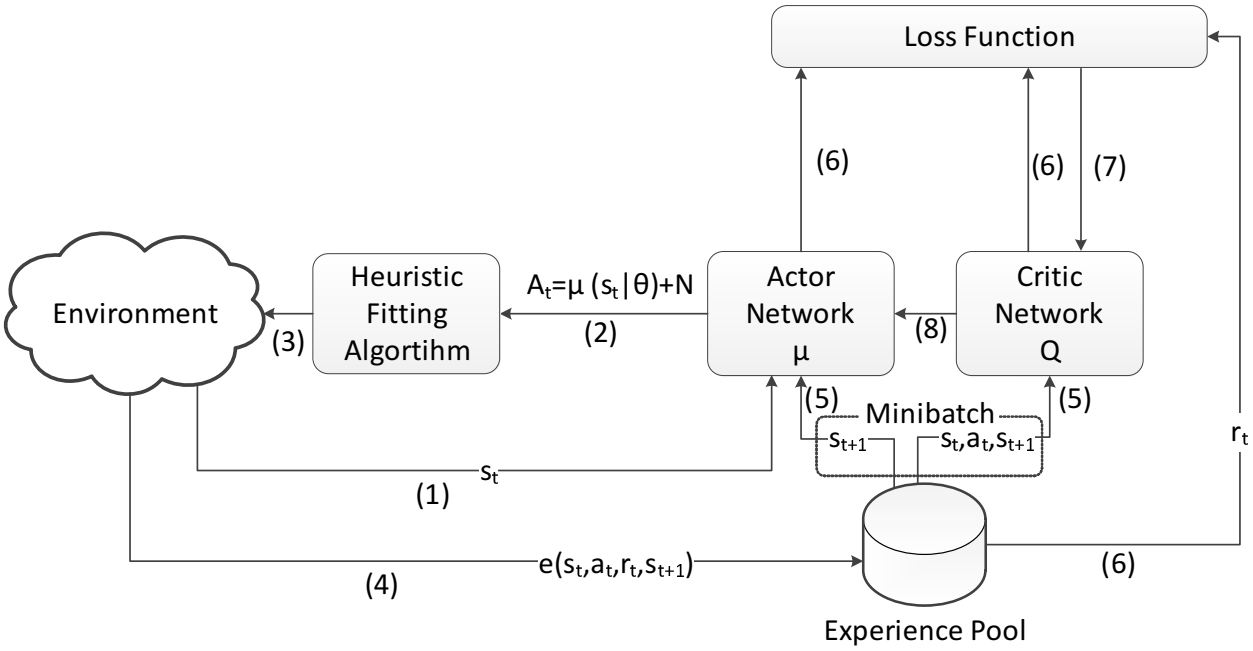

Fig. 2: DDPG–HFA

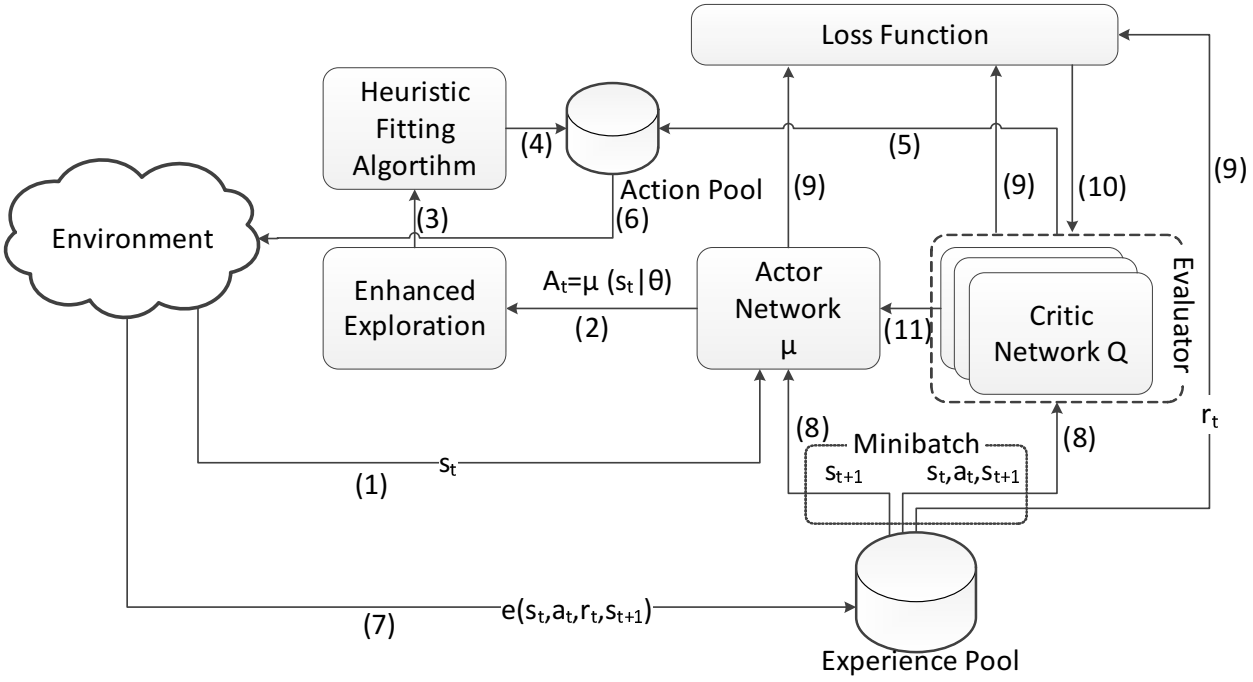

Fig. 3: E^2D^2PG

Algorithm 1: Heuristic Fitting Algorithm

Input: $\tilde{\mathbf{a}}_t$
Output: $\Phi_n^{n'}$,$\Phi_l^{l'}$
$\tilde{\mathbf{a}}_t = \left[\tilde{\mathbf{a}}_t^{n'}, \tilde{\mathbf{w}}_t^{l'}\right]$;
foreach n' **do**
    Sort $\tilde{\mathbf{a}}_t^{n'}$ in descending order;
    foreach $n_0 \in \tilde{\mathbf{a}}_t^{n'}$ **do**
        if $h_{n',k} \leq r_{n_0,k}, \forall k$ **then**
            $\Phi_{n_0}^{n'} = 1$;
            $\Phi_n^{n'} = 0, \forall n \neq n_0$;
            Update remaining resources:
            $r_{n,k} = r_{n,k} - h_{n',k}, \forall k$;
            break;
foreach l' **do**
    $n' \leftarrow \texttt{Source}\,(l')$;
    $m' \leftarrow \texttt{Destination}\,(l')$;
    $n \leftarrow \arg\max_n \Phi_n^{n'}$;
    $m \leftarrow \arg\max_n \Phi_n^{m'}$;
    Load weights of substrate links from $\tilde{\mathbf{w}}_t^{l'}$;
    foreach l **do**
        if $r_{l,bw} < h_{l',bw}$ **then**
            $\tilde{w}_t^{l,l'} \leftarrow 1$;
    Substrate path $P \leftarrow \texttt{Dijkstra}\,(n, m)$;
    Based on P, set $\Phi_l^{l'}$;

To create V noisy actions, the DRL agent initializes V Ornstein–Uhlenbeck (OU) processes. The noisy action v, $\hat{\mathbf{a}}_{t,v}$, is the combination of the proto-action with v^{th} OU process.

$$\hat{\mathbf{a}}_{t,v} = \tilde{\mathbf{a}}_t + \mathcal{N}_v(t), \tag{13}$$

where the proto action $\tilde{\mathbf{a}}_t$ which can be determined by the actor network. It means $\tilde{\mathbf{a}}_t = \mu\left(\mathbf{s}_t|\theta^\mu\right)$. By considering multiple noisy actions, the action space can be explored better.

2) Heuristic Fitting Algorithm: The noisy actions obtained in the previous step will be processed by HFA so as to find a set of actions which satisfy the requirement of VNFs and the bandwidth requirements of VLs. Then, these actions are evaluated by an evaluator which will be described in the following Section.

3) Evaluator: The evaluator is responsible for assessing the quality of the solution. In [31], the critic plays the role of an evaluator in order to select the best solution. Consequently, the quality of the critic will impact the quality of selection process. To improve the quality of the critic, one may enable parallel learning with multiple worker methods as proposed in [46], [47]. However, these methods need a number of independent environments which may not be always available. In VNF-FG problems, the environment is the substrate networks; therefore, it is impossible to create multiple environments for parallel learning.

In this paper, we propose a multiple critic network (MCN) method so as to enhance the learning process of the DRL agent. The evaluator comprises multiple critic networks which are initialized randomly. At time-step t, the action i has the Q-value $q_{t,i,j}$ computed by the critic network j and the mean of Q-values from all critic networks $\bar{q}_{t,i} = \frac{1}{K}\sum_{j=1}^{K} q_{t,i,j}$. At time-step t, the quality of critic networks may be quite different due to the arbitrary initialization. Consequently, the means of Q-values are utilized to assess the quality of actions. The action with the greatest mean Q-values is selected so as to execute.

It is expected that the actor network will be updated with the gradients of the best critic network among MCNs. A critic network, as its definition, provides an approximation of the Q-values. The loss indicates the difference between the approximation and the target value. Consequently, we adopt the loss as the metric to evaluate the quality of a critic network. The lowest loss critic network will be selected to update the actor network.

Alg.2 describes the proposed $\texttt{E}^2\texttt{D}^2\texttt{PG}$. In lines 1 and 2, K critic networks, the actor networks, K target critic networks, and the target actor network are initialized using Glorot normal initializer as discussed in [43]. 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 `HFA` (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 14 to line 16). Then, the loss of each critic networks is identified (from line 18 to line 20). The lowest loss critic network will be selected to update the actor network (from line 21 to line 23). Finally, the target networks are updated (from line 24 to line 26).

VI. Neural network architectures

For both actor and critic networks, we use M fully connected layers. A large M requires more computational efforts. Meanwhile, the small M may impact negatively to the performance since it is unable to extract all features from the states to identify appropriate actions.

To identify the best value of M, we evaluate the performance of the DRL agent with different values of M. Fig. 4 presents the performance of acceptance ratio under variation of M. A VNF-FG is accepted when all VNFs and VLs are deployed and its requirements (computing resources, bandwidth, QoS, etc.) are satisfied. The number of VNF-FGs is 5 and the topology is BtEurope [48]. It is shown in Fig. 4 that larger M has the better acceptance ratio and more stable. After 20 episodes, a DRL agent with $M \geq 6$ is able to obtain the acceptance ratio over 60%. Moreover, DRL agent with $M = 6$ obtains similar performance as other high M DRL agents ($M = 8, M = 10$)

Algorithm 2: Enhanced Exploration DDPG

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 V random process $\mathcal{N}_v, v=1,...,V$

  Receive initial observation state s_1

  foreach *t = 1,...,T* **do**

    Generate actions $\hat{\mathbf{a}}_{t,v}=\mu\left(\mathbf{s}_t|\theta^{\mu}\right)+\mathcal{N}_v, v=1,...,V$ based on the current policy and exploration noise k

    foreach *action* $\hat{\mathbf{a}}_{t,v}$ **do**

      $\mathbf{\Phi}^{n'}, \mathbf{\Phi}^{l'} \leftarrow \mathtt{HFA}\left(\hat{\mathbf{a}}_{t,v}\right)$;

      foreach *critic network* k **do**

        Compute $q_{t,v,k}=Q_k\left(\mathbf{s}_t,\mathbf{a}_{t,v}^{*}|\theta^{Q_k}\right)$

      Compute average q value of action $\mathbf{a}_{t,v}^{*}$: $\mathbb{Q}_t \leftarrow \bar{q}_{t,v}=\frac{1}{K}\sum_{k=1}^{K} q_{t,v,k}$

    $v^{*} \leftarrow \arg\max \mathbb{Q}_t$

    Execute the deployment of $\mathbf{\Phi}^{n'}$ and $\mathbf{\Phi}^{l'}$ and observe reward r_t and observe new state $\mathbf{s}_{t+1}$

    Store transition $\left(\mathbf{s}_t,\mathbf{a}_{t,v^*}^{*},r_t,\mathbf{s}_{t+1}\right)$ in R

    Sample a random minibatch of 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:

    $\nabla_{\theta^{\mu}} J \approx \frac{1}{N}\sum_{i}\nabla_{\mathbf{a}} Q_{k^*}\left(\mathbf{s},\mathbf{a}|\theta^{Q_{k^*}}\right)|_{\mathbf{s}=\mathbf{s}_i,\mathbf{a}=\mu(\mathbf{s}_i)}\nabla_{\theta^{\mu}}\mu\left(\mathbf{s}|\theta^{\mu}\right)|_{\mathbf{s}_i}$

    Update the target networks:

    $\theta^{Q_k^{'}} \leftarrow \tau\theta^{Q_k}+(1-\tau)\,\theta^{Q_k^{'}}, k=1,...,K$

    $\theta^{\mu^{'}} \leftarrow \tau\theta^{\mu}+(1-\tau)\,\theta^{\mu^{'}}$

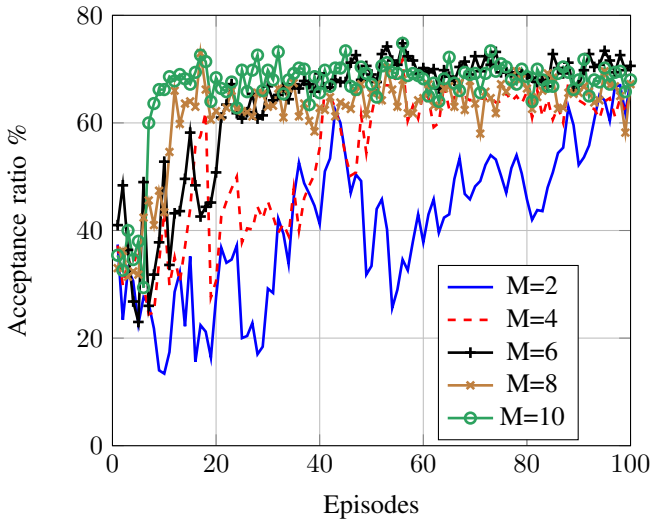

Fig. 4: Acceptance ratio of VNF-FGs under different values of M

in the long term. Thus, we select $M = 6$ for our implementation so as to achieve a balance between performance and computational complexity.

VII. Simulation results

To assess the performance of the proposed approach, we consider, for the first experiments, the network topology of BtEurope [48] with 24 nodes and 37 full-duplex links. Link capacity is one of following values randomly: 100 Mbps, 150 Mbps, 600 Mbps or 1 Gbps. The OMNeT++ discrete event simulator [49] (v5.4.1) was used to obtain the latency and packet loss rate to assess the quality of deployed VLs.

For each configuration, we execute 10 runs with different seeds of random generators. We adopt Erdős-Rényi model [50] to generate the graphs of VNF-FGs. The direction of a VL is selected randomly. 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 [51], 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. Although the VNF-FGs are frequently presented in directed trees, they are not necessarily directed trees [52], [53]. Therefore, we prefer to consider more generic services. Moreover, the maximum number of edges in a tree graph of $|\mathcal{N}^{'}|$ nodes is $|\mathcal{N}^{'}| - 1$. Meanwhile, the expected number of connections in a directed random graph is $p \times |\mathcal{N}^{'}| \times (|\mathcal{N}^{'}| - 1)/2$, where $|\mathcal{N}^{'}| \times (|\mathcal{N}^{'}| - 1)/2$ is the maximum number of available unidirectional links. With $p = 2.0 \times \frac{\log |\mathcal{N}^{'}|}{|\mathcal{N}^{'}|}$, the average number of unidirectional links of a random graph is $\log\left(|\mathcal{N}^{'}|\right)\left(|\mathcal{N}^{'}| - 1\right)$ which is greater than the number of edges of a tree graph, $|\mathcal{N}^{'} - 1|$, when $|\mathcal{N}^{'}| \geq 3$. Consequently, the tree graph is simpler than the random graph (lesser VLs); thus its action space is smaller than one of random graph (with the same number of VNFs). Each VNF-FG has from 5 to 11 VNFs. The requested resources of VNFs are normalized and distributed uniformly. The amounts of available resources of each substrate node are random in the range $(0.5, 2.0)$. Each virtual link arbitrarily requests a bandwidth in the range 1 Mbps to 40 Mbps, latency of 1 ms to 100 ms, and loss rate of 0.1% to 10%. There are 200 episodes for each run. In each episode, the number of steps is 100. In

each time-step, new configurations of VNF-FGs, as well as the amount of resources in the substrate network are generated. The number of VNF-FGs are unchanged in every time-step. At the end of each time-step, VNF-FGs are terminated and their performances are measured to compute the reward. The reward of each episode is the mean of rewards of every step in the episode.

The DRL agent is written in Python language with Keras library [54]. To run DRL agent and OMNet++ simulation, we have used a laptop of Intel i7-7920HQ and 32 GB RAM. The DRL agent is set up with the following parameters. Adam [55] is adopted to learn the neural network parameters. The learning rates of actor and critic networks are 10^{-4} and 10^{-3}, respectively. The discount factor γ is 1.0. The parameter of the target network is updated with the coefficient of $\tau = 0.001$. The batch size is 32. The number of units of fully-connected layers is 300. We adopt the Rectified Linear Unit (ReLU) activation [56] for dense layers except for the output of the actor network where the sigmoid activation is adopted so as to obtain the output in the range $(0, 1)$.

A. Number of critic networks

E^{2}D^{2}PG comes with an evaluator of MCNs. A greater number of MCNs requires more computational efforts; however, it may also return a better performance. In this section, we study the compromise between the number of critic networks and the performance. The acceptance ratio of VNF-FGs of the E^{2}D^{2}PG agents with 1, 10, and 20 critic networks are considered. The results are shown in Fig. 5. The E^{2}D^{2}PG agents with 10 and 20 critic networks have more stable performance than the E^{2}D^{2}PG agent with single critic network. It is because the action selection process of E^{2}D^{2}PG agent depends significantly to the quality of its single critic network. Meanwhile, the E^{2}D^{2}PG agents with more critic networks (10 and 20) have a better assessment of the potential actions by averaging the Q-values computed by critic networks. Besides, the best critic network will be selected so as to update the actor network; therefore enhancing the quality of the actor. The E^{2}D^{2}PG with 10 critic networks has similar performance with 20 critic network evaluator. Consequently, we use 10 critic networks for the evaluator.

B. Impact of HFA

We study the impact of HFA to the performance of DRL agent. Fig. 6 presents the comparison of the DDPG agent with and without HFA. DDPG computes the $a_t^{n,n'}$; then the substrate node with the greatest $a_t^{n,n'}, \forall n$ will be selected to host n'. The acceptance ratio of DDPG−HFA is much greater than that of DDPG because DDPG does not perform well when it comes to a large-scale discrete space [31]. Moreover, the selection of substrate node based on $a_t^{n,n'}$ does not guarantee sufficient resources for VNFs. DDPG−HFA has better performance thanks to HFA. The proto action after processing by HFA may meet the resource requirements of VNFs. Additionally, the substrate paths meet the bandwidth requirements of VLs. Consequently, the higher acceptance ratio can be achieved by DDPG−HFA.

C. E^{2}D^{2}PG performance

In this section, the performance of E^{2}D^{2}PG is studied. We compare the performance of E^{2}D^{2}PG with a heuristic approach, First-Fit-Dijkstra (FFD), and DDPG−HFA. FFD approach adopts First-Fit algorithm for allocating VNFs to the substrate network and Dijkstra algorithm (the shortest path in terms of the number of hops) to define the

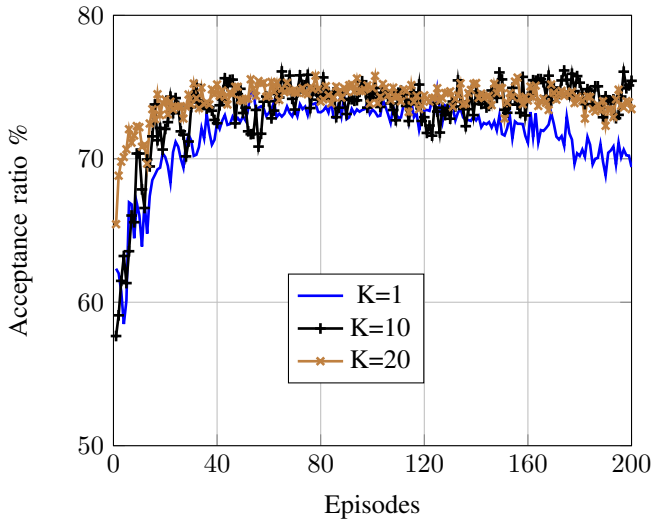

Fig. 5: Acceptance Ratio vs Number of critic networks

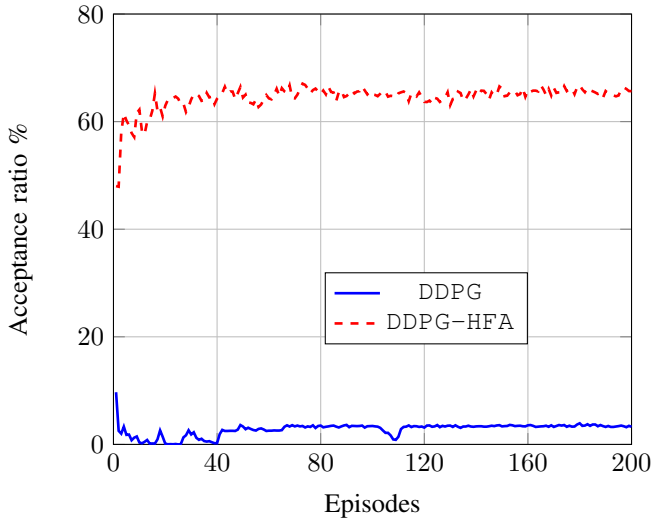

Fig. 6: Acceptance rate of DDPG and DDPG-HFA

substrate paths of VLs. The First-Fit algorithm is a heuristic algorithm which has been used to deploy VNFs. It has been used as one of the baselines to assess the performance in [57]–[59]. With given VNF allocation, Dijkstra algorithm could be adopted to determine the path to connect VNFs [60]. In DDPG-HFA and E^{2}D^{2}PG, the locations of VNFs and the substrate paths of VLs are determined by the neural networks and HFA. Consequently, the quality of the action generated by the DRL agent impacts significantly to the performance in DDPG-HFA and E^{2}D^{2}PG.

We consider three scenarios with 1, 3, and 5 VNF-FGs. The higher number of VNF-FGs means the resource requirements are greater and it is more difficult to deploy. In all scenarios, the acceptance ratios of DDPG-HFA and E^{2}D^{2}PG are greater than ones of FFD. Fig. 7 and 8 show the acceptance ratio and the percentage of deployed VLs when there is one VNF-FG. The acceptance ratio of VNFs is always 1, thus we do not show it here. The acceptance

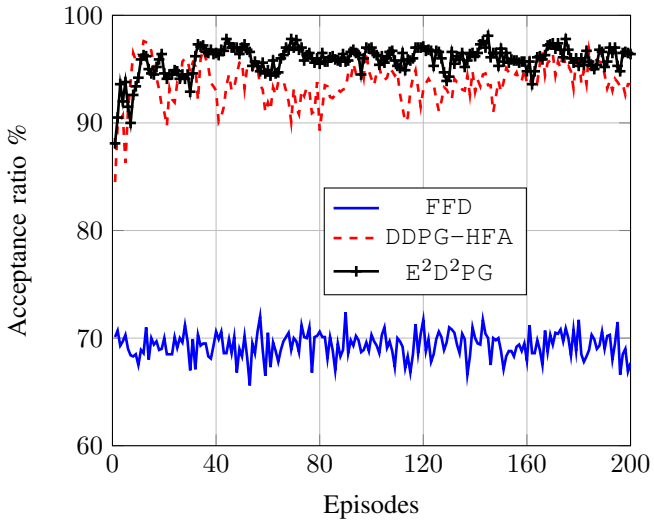

Fig. 7: Acceptance Ratio - 1 VNF-FG

ratio of DDPG−HFA is much greater than one of FFD (more than 20%). Moreover, DDPG−HFA takes less time to obtain a stable condition thanks to the simplicity in the resource requests. E^{2}D^{2}PG is better than DDPG−HFA in both the acceptance ratio and the percentage of deployed VLs. Fig. 9 and 10 shows the acceptance ratio, the percentage of deployed VLs when there are three VNF-FGs. DDPG−HFA and E^{2}D^{2}PG have better acceptance ratio than FFD. Although the percentages of successful deployed VLs of DDPG−HFA and FFD are similar, the acceptance ratio of DDPG−HFA is higher. Thanks to its intelligence, DDPG−HFA and E^{2}D^{2}PG are able to allocate resources to VLs smartly in order to optimize the acceptance ratio of VNF-FGs.

Fig. 11 shows the acceptance ratio of E^{2}D^{2}PG, DDPG−HFA, and FFD. FFD has the lowest acceptance ratio when the number of VNF-FGs is 5. Moreover, the acceptance ratio of FFD is not ameliorated over time. The acceptance ratio of DDPG−HFA is lower than FFD at the beginning since DDPG−HFA is exploring the action space at the beginning. However, the acceptance ratio of DDPG−HFA increases progressively and it outperforms FFD after a few episodes. E^{2}D^{2}PG outperforms both DDPG−HFA and FFD even from the beginning. The acceptance ratio of E^{2}D^{2}PG is up to 10% better than DDPG−HFA.

Besides the acceptance ratio, we also investigate the percentages of deployed VNFs and VLs. The results are shown in Fig. 12 and Fig. 13. The percentages of deployed VNFs and VLs are higher than the acceptance ratio since the good acceptance ratio is more difficult to achieve. For instance, a VNF-FG is not accepted even if there is only a VNF or a VL is not successfully deployed. Although the percentages of deployed VNFs of FFDis similar to these of E^{2}D^{2}PG, the percentages of deployed VLs of E^{2}D^{2}PG are significantly higher than ones of FFD(about 10%); consequently, the acceptance ratios of E^{2}D^{2}PG are much higher than one of FFD. The percentages of deployed VNFs and VLs of DDPG−HFA are lower than these of FFD; however, its acceptance ratio is higher than FFD. FFD , even though, is able to deploy more VNFs and VLs; its number of VNF-FGs which meets QoS requirements is lower than one of DDPG−HFA.

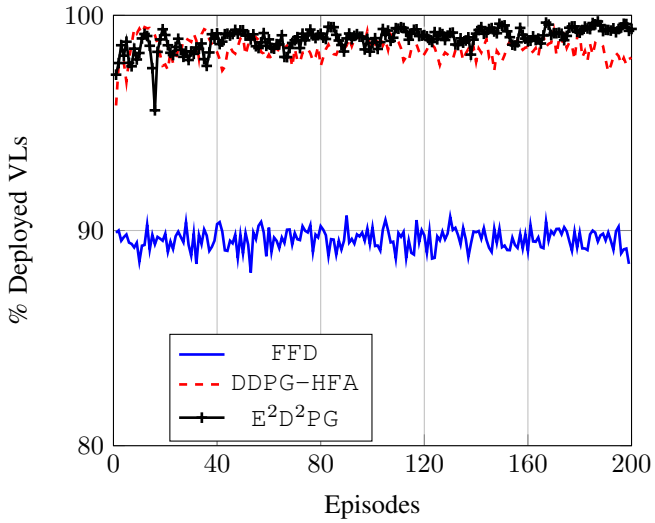

Fig. 8: Deployed VLs - 1 VNF-FG

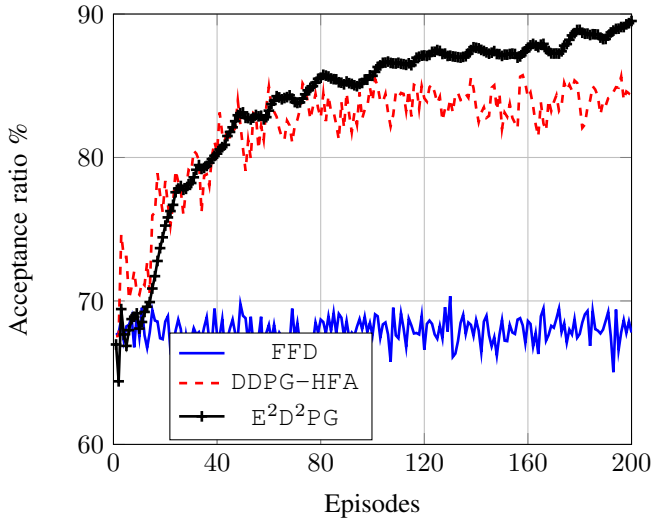

Fig. 9: Acceptance Ratio - 3 VNF-FGs

Next, we compare the acceptance ratio of E^{2}D^{2}PG and ILP-based solution. Note that the ILP-based solution solves 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 [61] and it is difficult to have an accurate model for these QoS metrics in multihop networks [37]. Existing studies formulated the VNF-FG embedding problem as an ILP problem (without considering QoS metrics, e.g. latency and loss rate) and solved it by proposing heuristic algorithms [45], [62]. The ILP-based solution is the optimal solution and requires extremely high computing resources. In return, its solution is the upper-bound of heuristic algorithms in [45], [62]; however, it is not the upper-bound in considered scenarios due to the missing of QoS constraints. Fig. 14 presents the acceptance ratio of ILP-based solution and E^{2}D^{2}PG. At the beginning, the acceptance ratio of E^{2}D^{2}PG is lower than one of ILP. However, it increases progressively

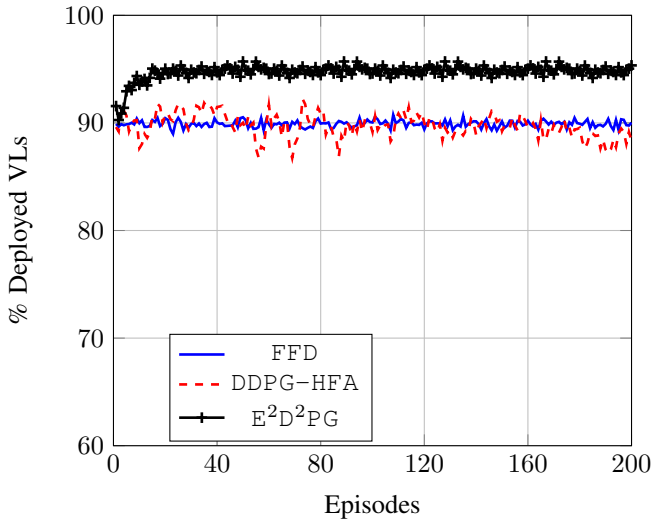

Fig. 10: Deployed VLs - 3 VNF-FGs

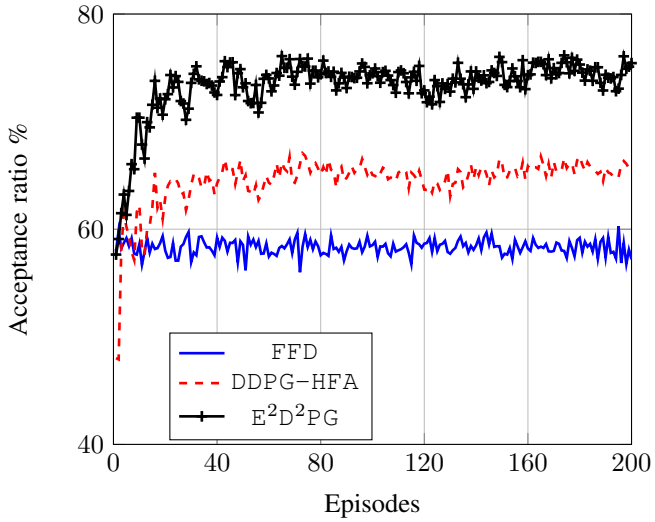

Fig. 11: Acceptance Ratio - 5 VNF-FGs

and outperforms ILP after a few episodes. This is because E^{2}D^{2}PG is able to learn the hidden patterns of the environment by interacting with the substrate network.

A more complicated substrate network is also considered to evaluate the performance of the proposed algorithms. The substrate network topology is Uninett[2], which has 74 nodes and 101 links [48]. This substrate network has about 3 times as much as nodes and links of BtEurope. The same neural networks and configurations have been used in these simulations. The acceptance ratio is shown in Fig. 15. Generally, the acceptance ratio is lower than one in BtEurope scenario. It is because a larger substrate network needs a deeper neural networks and more samples to

[2]http://www.topology-zoo.org/files/Uninett2010.gml

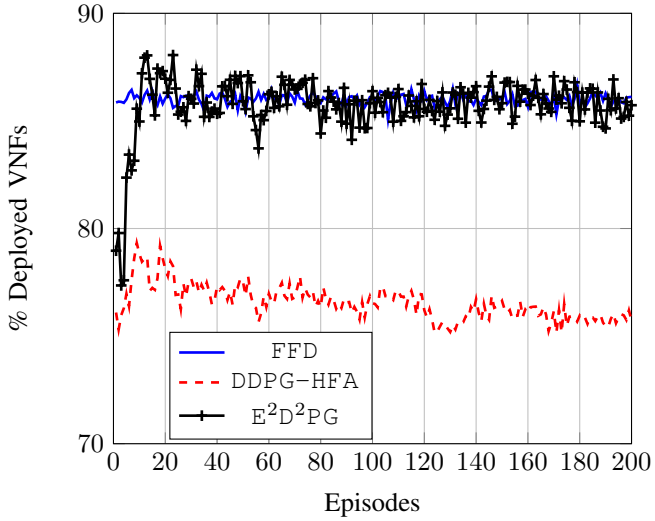

Fig. 12: Percentage of deployed VNFs - 5 VNF-FGs

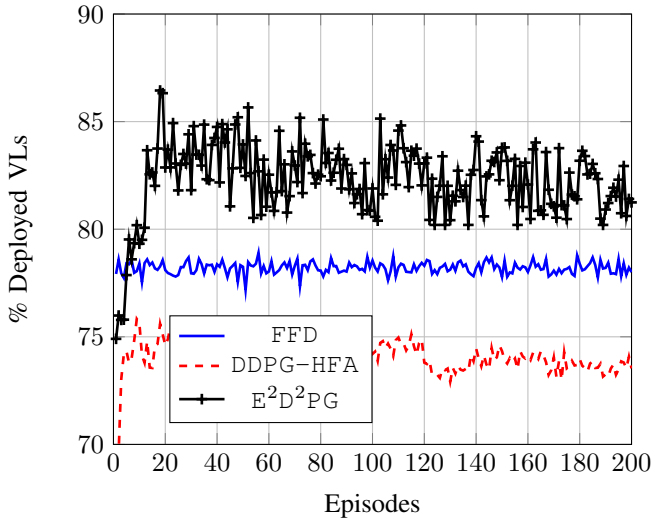

Fig. 13: Percentage of deployed VLs - 5 VNF-FGs

extract the hidden patterns from the environment (the substrate network). However, a deeper neural network leads to a longer training time and higher computational complexity. Without adding more layers, DDPG−HFA and E^{2}D^{2}PG still outperform FFDsignificantly. The acceptance ratio of E^{2}D^{2}PG can be up to twice of FFD. The acceptance ratio of E^{2}D^{2}PG is also 10% higher than DDPG−HFA.

VIII. Conclusions

NFV and service orchestration facilitate the management and the deployment of services in telecommunication networks. These services are able to be described in the VNF-FG; thus, the deployment of a service is, in fact, embedding the corresponding VNF-FG to a substrate network. The complexity of the VNF-FG embedding problem

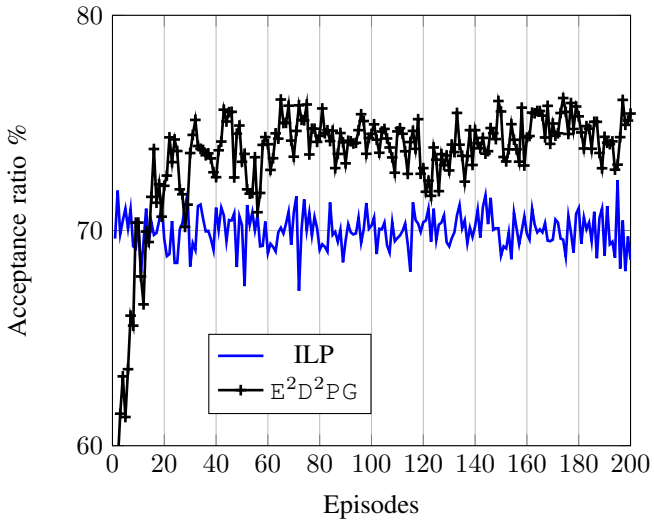

Fig. 14: Acceptance Ratio of ILP vs E^{2}D^{2}PG

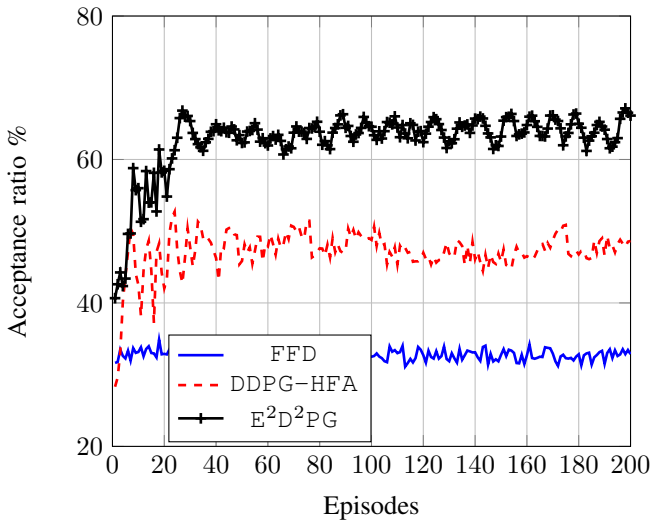

Fig. 15: Acceptance Ratio Uninett

and uncertainty of network metrics are the main challenges. Fortunately, emerging deep learning approaches unveil promising solutions to cope with these challenges, especially deep reinforcement learning approaches. DRL approaches can determine the near-optimal solution efficiently. Moreover, it is able to learn the characteristics of the network so as to ameliorate the quality of solutions. However, the state-of-the-art DRL algorithms are unable to deal with the problem with large-scale discrete space as VNF-FG embedding problem. This paper addressed the need of a light-weight but efficient DRL in order to deal with challenges in VNF-FG embedding problem. We modified the DDPG algorithm with a heuristic algorithm, named HFA, so as to enhance the quality provided by the actor network. Moreover, novel modules, i.e. evaluator and enhanced exploration, were added to form a new DRL aglorithm, named Enhanced Exploration DDPG (E^{2}D^{2}PG). The proposed algorithm enhances the exploration of

DRL agent. The results confirmed that the proposed algorithm outperforms DDPG algorithm as well as ILP-based solutions.

REFERENCES

[1] "Network Functions Virtualisation (NFV): Architectural framework," *ETSI GS NFV*, vol. 2, no. 2, p. V1, 2013.

[2] B. Yi, X. Wang, K. Li, S. k. Das, and M. Huang, "A comprehensive survey of Network Function Virtualization," *Computer Networks*, vol. 133, pp. 212 – 262, 2018.

[3] 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*, vol. 36, no. 3, pp. 529–541, March 2018.

[4] 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.

[5] M. Mechtri, C. Ghribi, and D. Zeghlache, "A Scalable Algorithm for the Placement of Service Function Chains," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 533–546, Sep. 2016.

[6] S. Khebbache, M. Hadji, and D. Zeghlache, "A multi-objective non-dominated sorting genetic algorithm for VNF chains placement," in *Proc. IEEE CCNC*, Jan 2018, pp. 1–4.

[7] 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.

[8] 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.

[9] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, "Approximation Algorithms for NP-hard Problems," D. S. Hochbaum, Ed. Boston, MA, USA: PWS Publishing Co., 1997, ch. Approximation Algorithms for Bin Packing: A Survey, pp. 46–93.

[10] A. Tomassilli, F. Giroire, N. Huin, and S. Pérennes, "Provably efficient algorithms for placement of service function chains with ordering constraints," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, April 2018, pp. 774–782.

[11] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions," in *Proc. IFIP/IEEE IM*, May 2015, pp. 98–106.

[12] 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.

[13] H. Ko, D. Suh, H. Baek, S. Pack, and J. Kwak, "Optimal placement of service function in service function chaining," in *Proc. ICUFN)*, July 2016, pp. 102–105.

[14] Q. Zhang, Y. Xiao, F. Liu, J. C. Lui, J. Guo, and T. Wang, "Joint optimization of chain placement and request scheduling for network function virtualization," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 731–741.

[15] A. Shpiner, Z. Haramaty, S. Eliad, V. Zdornov, B. Gafni, and E. Zahavi, "Dragonfly+: Low cost topology for scaling datacenters," in *2017 IEEE 3rd International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*, Feb 2017, pp. 1–8.

[16] A. Tomassilli, F. Giroire, N. Huin, and S. Pérennes, "Provably efficient algorithms for placement of service function chains with ordering constraints," in *Proc. IEEE INFOCOM*, April 2018, pp. 774–782.

[17] S. R. Chowdhury, R. Ahmed, N. Shahriar, A. Khan, R. Boutaba, J. Mitra, and L. Liu, "ReViNE: Reallocation of Virtual Network Embedding to eliminate substrate bottlenecks," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, May 2017, pp. 116–124.

[18] 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.

[19] N. Tastevin, M. Obadia, and M. Bouet, "A graph approach to placement of service functions chains," in *Proc. IFIP/IEEE IM*, May 2017, pp. 134–141.

[20] M. M. Tajiki, S. Salsano, L. Chiaraviglio, M. Shojafar, and B. Akbari, "Joint Energy Efficient and QoS-Aware Path Allocation and VNF Placement for Service Function Chaining," *IEEE Transactions on Network and Service Management*, vol. 16, no. 1, pp. 374–388, March 2019.

[21] J. Pei, P. Hong, K. Xue, and D. Li, "Efficiently embedding service function chains with dynamic virtual network function placement in geo-distributed cloud system," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–1, 2018.

[22] S. Kim, S. Park, Y. Kim, S. Kim, and K. Lee, "VNF-EQ: dynamic placement of virtual network functions for energy efficiency and QoS guarantee in NFV," *Cluster Computing*, vol. 20, no. 3, pp. 2107–2117, Sep 2017.

[23] J. Cao, Y. Zhang, W. An, X. Chen, J. Sun, and Y. Han, "VNF-FG design and VNF placement for 5G mobile networks," *Science China Information Sciences*, vol. 60, no. 4, p. 040302, Mar 2017.

[24] M. Otokura, K. Leibnitz, Y. Koizumi, D. Kominami, T. Shimokawa, and M. Murata, "Application of evolutionary mechanism to dynamic virtual network function placement," in *Proc. IEEE ICNP*, Nov 2016, pp. 1–6.

[25] M. C. Luizelli, W. L. da Costa Cordeiro, L. S. Buriol, and L. P. Gaspary, "A fix-and-optimize approach for efficient and large scale virtual network function placement and chaining," *Computer Communications*, vol. 102, pp. 67 – 77, 2017.

[26] T. Bäck, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. New York, NY, USA: Oxford University Press, Inc., 1996.

[27] S. Singh, A. Okun, and A. Jackson, "Learning to play Go from scratch," *Nature*, vol. 550, pp. 336–337, 10 2017.

[28] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *CoRR*, vol. abs/1611.09940, 2016.

[29] T. A. Q. Pham, Y. Hadjadj-Aoul, and A. Outtagarts, "Deep Reinforcement Learning Based QoS-Aware Routing in Knowledge-Defined Networking," in *Proc. EAI QShine*, Dec. 2019, pp. 14–26.

[30] 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.

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

[32] F. Carpio, S. Dhahri, and A. Jukan, "VNF placement with replication for Load balancing in NFV networks," in *IEEE ICC*, May 2017, pp. 1–6.

[33] F. Carpio, W. Bziuk, and A. Jukan, "Replication of Virtual Network Functions: Optimizing link utilization and resource costs," in *MIPRO*, May 2017, pp. 521–526.

[34] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba, "Elastic virtual network function placement," in *CloudNet*, Oct 2015, pp. 255–260.

[35] 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.

[36] K. Rusek, J. Suárez-Varela, A. Mestres, P. Barlet-Ros, and A. Cabellos-Aparicio, "Unveiling the potential of graph neural networks for network modeling and optimization in SDN," *CoRR*, vol. abs/1901.08113, 2019.

[37] 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.

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

[39] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992.

[40] J. Tsitsiklis and B. Van Roy, "An analysis of temporal-difference learning with function approximation (technical report lids-p-2322)," *Laboratory for Information and Decision Systems*, 1996.

[41] 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.

[42] S. Adam, L. Busoniu, and R. Babuska, "Experience replay for real-time reinforcement learning control," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 2, pp. 201–212, March 2012.

[43] 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.

[44] 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.

[45] 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.

[46] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[47] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. TB, A. Muldal, N. Heess, and T. P. Lillicrap, "Distributed distributional deterministic policy gradients," *CoRR*, vol. abs/1804.08617, 2018.

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

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

[50] P. Erdős and A. Rényi, "On random graphs I," *Publ. Math. Debrecen*, vol. 6, pp. 290–297, 1959.

[51] ——, "On the evolution of random graphs," *Publ. Math. Inst. Hungar. Acad. Sci*, vol. 5, pp. 17–61, 1960.

[52] "Network functions virtualisation (nfv); management and orchestration," ETSI, DGS/NFV-MAN001 Standard, Dec. 2014.

[53] "Service function chaining (sfc) architecture," IETF, ISSN: 2070-1721, RFC 7665 Standard, Oct. 2015.

[54] K. team, "Keras," https://github.com/keras-team/keras, 2015.

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

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

[57] L. Yala, P. A. Frangoudis, G. Lucarelli, and A. Ksentini, "Cost and availability aware resource allocation and virtual function placement for cdnaas provision," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1334–1348, Dec 2018.

[58] L. Ochoa-Aday, C. Cervelló-Pastor, A. Fernández-Fernández, and P. Grosso, "An online algorithm for dynamic nfv placement in cloud-based autonomous response networks," *Symmetry*, vol. 10, no. 5, 2018.

[59] A. Marotta, E. Zola, F. D'Andreagiovanni, and A. Kassler, "A fast robust optimization-based heuristic for the deployment of green virtual network functions," *Journal of Network and Computer Applications*, vol. 95, pp. 42 – 53, 2017.

[60] 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.

[61] 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.

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