On the Robustness of Controlled Deep Reinforcement Learning for Slice Placement

Jose Jurandir Alves Esteves (corresponding author) *†, Amina Boubendir *, Fabrice Guillemin * and Pierre Sens †
*Orange Labs, France †Sorbonne Université / CNRS / Inria, LIP6, France
{josejurandir.alvesesteves, amina.boubendir, fabrice.guillemin}@orange.com, pierre.sens@lip6.fr

***Abstract*—The evaluation of the impact of using Machine Learning in the management of softwarized networks is considered in multiple research works. In this paper, we propose to evaluate the robustness of online learning for optimal network slice placement. A major assumption in this study is to consider that slice request arrivals are non-stationary. We precisely simulate unpredictable network load variations and compare two Deep Reinforcement Learning (DRL) algorithms: a pure DRL-based algorithm and a heuristically controlled DRL as a hybrid DRL-heuristic algorithm, in order to assess the impact of these unpredictable changes of traffic load on the algorithms performance. We conduct extensive simulations of a large-scale operator infrastructure. The evaluation results show that the proposed hybrid DRL-heuristic approach is more robust and reliable than pure DRL in real network scenarios.**

***Index Terms*—Network Slicing, Placement, Optimization, Deep Reinforcement Learning, Robustness, Reliability.**

I. Introduction

THE promise of network Slicing is to enable a high level of customization of network services in future networks (5G and beyond) leveraged by virtualization and software defined networking techniques. These key enablers transform telecommunications networks into programmable platforms capable of offering virtual networks enriched by Virtual Network Functions (VNFs) and IT resources tailored to the specific needs of certain customers (e.g., companies) or vertical markets (automotive, e-health, etc.) [1], [2]. From an optimization theory perspective, the Network Slice Placement problem can be viewed as a specific case of Virtual Network Embedding (VNE) or VNF Forwarding Graph Embedding (VNF-FGE) problems [3], [4].

In this context, it is generally possible to formulate Integer Linear Programming (ILP) problems [5], which however turn out to be $\mathcal{NP}$-hard [6] with very long execution time. This point is all the more crucial as network slices are expected to share resources and coexist in a large and distributed infrastructure. Moreover, slices have a wide range of requirements in terms of resources, quality objectives and lifetime. This brings additional complexity with regard to placement as optimization algorithms need to be highly scalable with low response time even under varying network conditions.

J.J. Alves Esteves is with Orange Labs, 92320 Chatilon, France and also with LIP6 – Inria, Sorbonne Univ., 75005 Paris, France (e-mail: josejurandir.alvesesteves@orange.com).

A. Boubendir and F. Guillemin are with Orange Labs, 92320 Chatillon, France (e-mail: firstname.name@orange.com).

P. Sens is with LIP6 – Inria, Sorbonne Univ., CNRS, 75005 Paris, France (e-mail: pierre.sens@lip6.fr).

As an alternative to optimization techniques and the development of heuristic methods, Deep Reinforcement Learning (DRL) has recently been used in the context of both VNE and Network Slice Placement [7]–[12]. DRL techniques are considered as very promising since they allow, at least theoretically, the determination of optimal decision policies only based on experience [13]. However, from a practical point of view, especially in the context of non-stationary environments, ensuring that a DRL agent converges to an optimal policy is still challenging as shown by the evaluation results in [14].

As a matter of fact, when the environment is continually changing, the algorithm has trouble in using the acquired knowledge to find optimal solutions. The use of a DRL algorithm online can then become impractical. In fact, most of existing works based on DRL to solve the Network Slice Placement or VNE problem assume a stationary environment, i.e., with constant network load. However, traffic conditions in real networks are basically non stationary with daily and weekly variations and subject to drastic changes (e.g., traffic storm due to an unpredictable event).

To cope with traffic changes, this paper proposes a hybrid DRL-heuristic strategy called Heuristically Assisted DRL (HA-DRL) [14]. We applied in [15] this strategy in an online learning scenario with periodically varying network load conditions to show how this strategy can be used to accelerate and stabilize the convergence of DRL techniques in this type of non-stationary environment. As a follow-up of theses two studies, we focus in the present paper on a different non stationary scenario with stair-stepped network load changes. The goal of the paper is to evaluate and show the robustness of the proposed strategy method in the case of sudden and stair-stepped traffic changes.

The contributions of the present paper are threefold:

1) We propose a network load model to describe network slice demand and adapt it to unpredictable network load changes;
2) We propose a framework combining Advantage Actor Critic and a Graph Convolutional Network (GCN) for conceiving DRL-based algorithms adapted to the non-stationary case;
3) We show how the use of a heuristic function can control the DRL learning capability and improve its robustness to unpredictable network load changes.

The organization of this paper is as follows: In Section II, we review related work. In Section III, we describe the Network Slice Placement problem modeling. The learning framework

for slice placement optimization is described in Section IV. The adaptation of the pure DRL approaches and its control by using a heuristic is introduced in Section IV-B. The experiments and evaluation results are presented in Section V, while conclusions and perspectives are presented in Section VI.

II. Related Work Analysis

We provide in Section II-A a succinct review of the existing DRL-based approaches for network slice placement. The interested reader may refer to [14], [15] for a more detailed and comprehensive discussion. In Section II-B we discuss recent works on robust slice placement algorithms.

A. On DRL-based Approaches for Slice Placement

DRL has been recently applied to solving network slice placement and VNE problems. We divide these works into two categories on the basis of their algorithmic aspects: 1) pure DRL approaches [7]–[12], in which only the knowledge acquired by the learning agent via training is used as a basis for taking placement decisions; and 2) hybrid DRL-heuristic approaches [14], [16], [17], in which the placement decision computation is assisted by a heuristic method.

The use of heuristics aims at increasing the reliability of DRL algorithms. However, most of these works are based on the assumption that the network load is static, i.e., slice arrivals occurs at a constant rate. To the best of our knowledge, the work we proposed in [15] is the first attempt to evaluate an online DRL-based approach in a non-stationary network load scenario whereas [18] only considers offline learning.

In addition, in both [15] and [18], it is assumed that the network load has periodic fluctuations. In the present paper we study the behavior of the algorithms proposed in [15] in case of an unpredictable network load disruption.

B. On Robustness of Slice Placement Approaches

The term robustness has different meanings depending on the field of application. In Robust Optimization (RO), robustness is related to the decision/solution itself. It is the capability of the algorithm solution of coping with the worst case without losing feasibility [19]. In Machine Learning (ML), especially in Deep Learning (DL), robustness is related to the learned model. It is the property of the model (i.e., Deep Neural Network (DNN)) that determines its integrity under varying operation conditions [20].

The authors of [21] are the first ones to discuss robustness in the DRL context. They propose to use Genetic Algorithm to improve the robustness of a self-driving car application. Robustness is considered as the capacity of sustaining a high accuracy on image classification even when perceived images change and it is measured by Neuron Coverage (NC), i.e.,the ratio of the activated neurons in the DNN.

There are only a few recent works on the robustness of slice placement procedures, most of them on RO [22]–[25]. These works answer a question different from the one we are investigating in this paper as they evaluate the robustness of the decision whereas we want to evaluate the robustness of the learning process. Despite their originality, the above approaches present some drawbacks, such as the lack of scalability of ILP, the sub-optimality of heuristic solutions, the fact that they consider offline optimization in which all slices to be placed are known in advance, and the fact that they are single objective optimization approaches, mainly focusing on energy consumption minimization.

In the present work, we propose to rely on a DRL-based approach in order to overcome ILP and heuristic drawbacks and consider multiple-optimization objectives. To the best of our knowledge, paper [26] is the only one to have proposed a DRL-based approach for slice placement and evaluated the learning robustness. However, the authors focus on evaluating the robustness of the DRL approach against random topology changes (e.g., node failures or deploying new nodes in the network topology). Here, we focus on evaluating robustness against network load unpredictable variations. It seems that our work is the first to perform such an evaluation.

III. Network Slice Placement Optimization Problem

We present in this section the various elements composing the model for slice placement. Slices are placed on a substrate network, referred to as Physical Substrate Network (PSN) and described in Section III-A. Slices give rise to Network Slice Placement Requests (Section III-B), generating a network load defined in Section III-C. The optimization problem is formulated in Section III-D.

A. Physical Substrate Network Modeling

The Physical Substrate PSN is composed of the infrastructure resources, namely IT resources (CPU, RAM, disk, etc.) needed for supporting the Virtual Network Functions (VNFs) of network slices together with the transport network, in particular Virual Links (VLs) for interconnecting the VNFs of slices. Fig. 1 shows the PSN structure, which is divided into three main components: the Virtualized Infrastructure (VI) corresponding to IT resources, the Access Network (AN), and the Transport Network (TN). The Virtual Infrastructure (VI) hosting IT resources is the set of Data Centers (DCs) interconnected by network elements (switches and routers). We assume that data centers are distributed in Points of Presence (PoP) or centralized (e.g., in a big cloud platform).

As in [27], we define three types of DCs with different capacities: Edge Data Centers (EDCs) close to end users but with small resources capacities, Core Data Centers (CDCs) as regional DCs with medium resource capacities, and Central Cloud Platforms (CCPs) as national DCs with big resource capacities. We consider that slices are rooted so as to take into account the location of those users of a slice. We thus introduce an Access Network (AN) representing User Access Points (UAPs) such as Wi-Fi APs, antennas of cellular networks, etc. and Access Links. Users access slices via one UAP, which may change during the life time of a communication by a user (e.g., because of mobility). The Transport Network (TN) is the set of routers and transmission links needed to interconnect the different DCs and the UAPs.

The complete PSN is modeled as a weighted undirected graph $G_s = (N, L)$ with parameters described in Table I, where N is the set of physical nodes in the PSN, and $L \subset \{(a, b) \in N \times N : a \neq b\}$ refers to a set of substrate links. Each node has a type in the set {UAP, router, switch, server}. The available CPU and RAM capacities on each node are defined as $cap_n^{cpu} \in \mathbb{R}$, $cap_n^{ram} \in \mathbb{R}$ for all $n \in N$, respectively. The available bandwidth on the links are defined as $cap_{(a,b)}^{bw} \in \mathbb{R}$ for all $(a, b) \in L$.

TABLE I
PSN PARAMETERS

Parameter	*Description*
$G_s = (N, L)$	PSN graph
N	Network nodes
$S \subset N$	Set of servers
DC	Set of data centers
$S_{dc} \subset S, \forall dc \in DC$	Set of servers in data center dc
$SW_{dc}, \forall dc \in DC$	Switch of of data center dc
$L = \{(a, b) \in N \times N \wedge a \neq b\}$	Set of physical links
$cap_{(a,b)}^{bw} \in \mathbb{R}, \forall (a, b) \in L$	Bandwidth capacity of link (a, b)
$cap_s^{cpu} \in \mathbb{R}, \forall s \in S$	available CPU capacity on server s
$M_s^{cpu} \in \mathbb{R}, \forall s \in S$	maximum CPU capacity of server s
$cap_s^{ram} \in \mathbb{R}, \forall s \in S$	available RAM capacity on server s
$M_s^{ram} \in \mathbb{R}, \forall s \in S$	maximum RAM capacity of server s
$M_s^{bw} \in \mathbb{R}, \forall s \in S$	maximum outgoing bandwidth from s

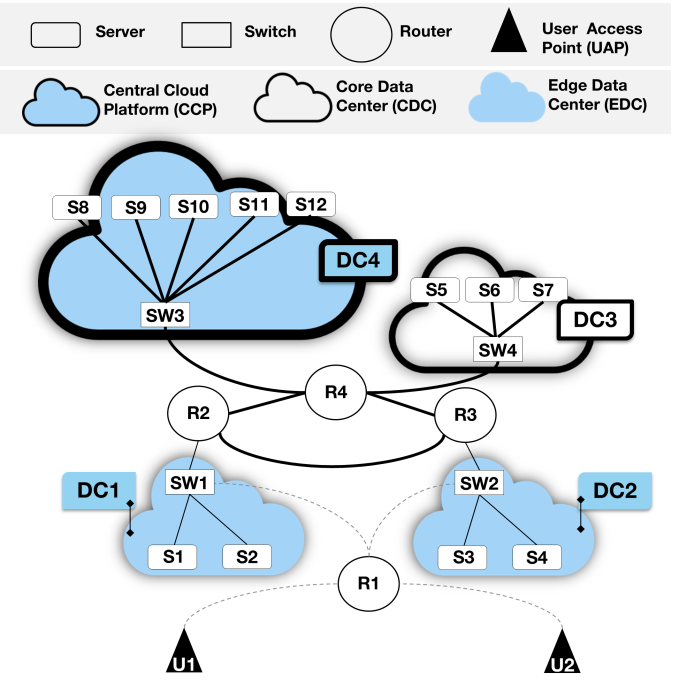


Fig. 1. Physical Substrate Network example.

B. Network Slice Placement Requests Modeling

We consider that a slice is a chain of VNFs to be placed and connected over the PSN. The VNFs of a slice are grouped into a request, namely a Network Slice Placement Request (NSPR), which has to be placed on the PSN. Note that in this paper, we only consider the computation of the initial placement and we consider that an NSPR has a fixed structure. The study of the reconfiguration of a slice after the initial placement (for instance via the migration of some VNF from one data center to another) is out of the scope of this paper. An NSPR is represented as a weighted undirected graph $G_v = (V, E)$, with parameters described in Table II, where V is the set of VNFs in the NSPR and $E \subset \{(\bar{a}, \bar{b}) \in V \times V \wedge \bar{a} \neq \bar{b}\}$ is a set of VLs interconnecting the VNFs of the slice . The CPU and RAM requirements of each VNF of an NSPR are defined as $req_v^{cpu} \in \mathbb{R}$ and $req_v^{ram} \in \mathbb{R}$ for all $v \in V$, respectively. The bandwidth required by each VL in an NSPR is given by $req_{(\bar{a},\bar{b})}^{bw} \in \mathbb{R}$ for all $(\bar{a}, \bar{b}) \in E$.

We consider the existence of different NSPR classes characterizing different levels of resources requirements, lifespan and arrival rate at described in Section III-C.

TABLE II
NSPR PARAMETERS

Parameter	*Description*
$G_v = (V, E)$	NSPR graph
V	Set of VNFs of the NSPR
$E = \{(\bar{a}, \bar{b}) \in N \times N \wedge \bar{a} \neq \bar{b}\}$	Set of VLs of the NSPR
$req_v^{cpu} \in \mathbb{R}$	CPU requirement of VNF v
$req_v^{ram} \in \mathbb{R}$	RAM requirement of VNF v
$req_{(\bar{a},\bar{b})}^{bw} \in \mathbb{R}$	Bandwidth requirement of VL $(\bar{a}, \bar{b})$

C. Network Load Modeling

The Network Load model allows us to control the percentage of the total network resources capacity being used at a specific instant. Let J be the set of resources in the network (i.e., CPU, RAM, bandwidth). Let $\mathcal{K} \subset \mathbb{N}$ be the set of NSPR classes. We compute the load generated by arrivals of NSPRs of class $k \in \mathcal{K}$ for resource j in J as in [28]:

$$\rho_j^k = \frac{1}{C_j} \frac{\lambda^k}{\mu^k} A_j^k, \tag{1}$$

where C_j is the total capacity of resource j, A_j^k is the number of resource units requested by an NSPR of class k, λ^k is the average arrival rate of NSPRs of class k, and $1/\mu^k$ is the average lifetime of an NSPR of class k.

We define the global load ρ_j for resource j as the sum

$$\rho_j = \sum_{k \in \mathcal{K}} \rho_j^k \tag{2}$$

If $0 \leq \rho_j \leq 1$, the system is not overloaded for resource j; otherwise, the system is under overload conditions and the rejection of NSPRs may be high.

D. Network Slice Placement Optimization Problem Statement

The Network Slice Placement optimization problem is stated as follows (see [14] for more details):

- *Given:* an NSPR graph $G_v = (V, E)$ and a PSN graph $G_s = (N, L)$,
- *Find:* a mapping $G_v \rightarrow \bar{G}_s = (\bar{N}, \bar{L})$, $\bar{N} \subset N$, $\bar{L} \subset L$,
- *Subject to:* the VNF CPU requirements $req_v^{cpu}, \forall v \in V$, the VNF RAM requirements $req_v^{ram}, \forall v \in V$, the

VLs bandwidth requirements $req^{bw}_{(\bar{a},\bar{b})}, \forall(\bar{a},\bar{b}) \in E$, the server CPU available capacity $cap^{cpu}_s, \forall s \in S$, the server RAM available capacity $cap^{ram}_s, \forall s \in S$, the physical link bandwidth available capacity $cap^{bw}_{(a,b)}, \forall(a,b) \in L$.

- *Objective:* maximize the network slice placement request acceptance ratio, minimize the total resource consumption and maximize load balancing.

E. P2C Heuristic Principles

A complete mathematical formulation of the network slice placement introduced in Section III-D problem can be found in [14] and we have proposed in [29] a heuristic to efficiently solve it. This heuristic is based on the P2C principle [30], which states in the present context that considering two possible data centers chosen "randomly" instead of only one brings exponential improvement of the solution quality. The pseudo code of the proposed heuristic is given illustrated in Algorithm 1. It is a greedy algorithm such that for each VNF $\bar{b} \in V$:

1) Randomly select 2 candidate servers $s_1, s_2 \in S$;
2) Evaluate the resource consumption when placing $\bar{b}$ in s_1 and s_2 and place $\bar{b}$ on the best server;
3) Map the VLs $(\bar{a},\bar{b}) \in E$ associated to $\bar{b}$.

This heuristic contains the limitations of all heuristic approaches: the lack of flexibility due to manual feature design, the difficulties to handle multiple optimization criteria, and the sub-optimality of the provided solutions. But it also yields low execution time and good load balancing; it was shown in [29] that the heuristic outperforms two ILP based algorithms. We elaborate in the next section on these properties of the heuristic to propose our HA-DRL approach.

Algorithm 1: Heuristic for Network Slice Placement Optimization using Power of two Choices (P2C).

```
Data: NSPR, PSN
Starting by the root VNF of the NSPR
while not at end of the NSPR do
    Calculate the set of candidate servers to place the current VNF of the NSPR
    if There is available candidate servers then
        Choose randomly two candidate servers to place the VNF
        Calculate resource consumption cost for placing the VNF in the both servers considering VNF placement and VL mapping
        Place the current VNF in the server providing lower resource consumption cost and map VL in the associated path
    else
        Backtrack to initial PSN state and reject current NSPR
        return
    Move to the next VNF of the NSPR
Accept current NSPR
return
```

IV. Learning framework for Network Slice Placement Optimization

We describe in this section the DRL-based approach used to solve the optimization problem formulated in Section III. As mentioned above, we adopt the same approach as in [14] but we focus in this paper on evaluating the performance when an unpredictable network load change occurs.

A. Learning framework

Figure 2 presents an overview of the DRL framework. The state contains the features of the PSN and NSPR to be placed. A valid action is, for a given NSPR graph $G_v = (V, E)$, a subgraph of the PSN graph $\bar{G}_s \subset \bar{G}_s = (N, L)$ to place the NSPR that does not violate the problem constraints described in [14] Section III-D. The reward evaluates how good is the computed action with respect to the optimization objectives described in [14] Section III-D.

Deep Neural Networks (DNNs) are trained to calculate i) optimal actions for each state (i.e., placements with maximal rewards) and ii) the State-value function used in the learning process. In the following sections, we describe each element of this framework.

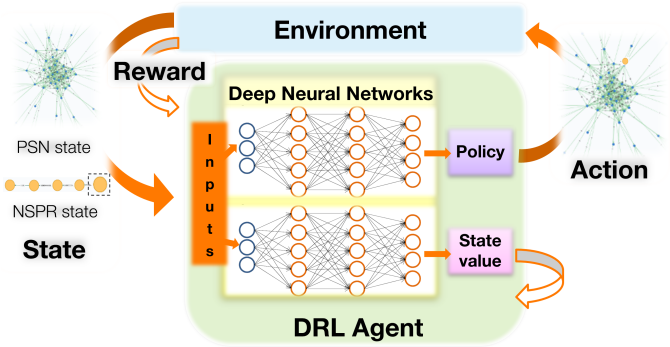


Fig. 2. DRL framework for Network Slice Placement Optimization

1) Policy: We reuse the framework introduced in [14]. We denote by $\mathcal{A}$ the set of possible actions (namely placing VNFs on nodes) and by $\mathcal{S}$ the set of all states. We adopt a sequential placement strategy so that we choose a node $n \in N$ where to place a specific VNF $v \in \{1, ..., |V|\}$. The VNFs are placed in ascending order, which means that the placement starts with the VNF $v = 1$ and ends for the VNF $v = |V|$.

We break then the process of placing one NSPR graph $G_v = (V, E)$ into a sequence of $|V|$ actions, one for each $v \in V$, instead of considering the one shot placement of G_v. The latter strategy would require the definition of the action as a subgraph of the PSN graph $G_s = (N, L)$ what would imply $|\mathcal{A}| = |SG|$, where SG is the set of all sub graphs of G_s, that grows exponentially with size of G_s. Note that with the sequential placement strategy $\mathcal{A} = N$, thus $|\mathcal{A}| \ll |SG|$. At each time step t, the DRL agent focuses on the placement of exactly one VNF $v \in V$ of the NSPR. The mapping of the virtual links associated to the VNF v to a physical path in the PSN is done by the shortest path algorithm.

At each time step t, given a state σ_t, the learning agent selects an action a with probability given by the Softmax distribution

$$\pi_\theta(a_t = a|\sigma_t) = \frac{e^{Z_\theta(\sigma_t,a),}}{\sum_{b\in N} e^{Z_\theta(\sigma_t,b)}}, \tag{3}$$

where the function $Z_\theta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ yields a real value for each state and action calculated by a DNN as detailed in Section IV-B1. The notation π_θ is used to indicate that policy depends on Z_θ. The control parameter θ represents the weights in the DNN.

2) State representation: As in [14], the **PSN state** is characterized by available server resources: $cap^{cpu} = \{cap_n^{cpu} : n \in N\}$, $cap^{ram} = \{cap_n^{ram} : n \in N\}$ and $cap^{bw} = \{cap_n^{bw} = \sum_{(n,b)\in L} cap_{(n,b)}^{bw} : n \in N\}$. In addition, we keep track of the placement of the outstanding NSPR (i.e., the one being placed) via the vector $\chi = \{\chi_n \in \{0, .., |V|\} : n \in N\}$, where χ_n is the number of VNFs of the outstanding NSPR placed on node n.

The **NSPR state** is a view of the current placement and is composed of four characteristics, three related to resource requirements (see Table II for the notation) of the current VNF v to be placed: req_v^{cpu}, req_v^{ram} and $req_v^{bw} = \sum_{(v,\bar{b})\in E} req_{(v,\bar{b})}^{bw}$, and $m_v = |V| - v + 1$ the number of VNFs of the outstanding NSPR still to be placed.

3) Reward function: We reuse the reward function introduced in [14]. We precisely consider

$$r_{t+1} = \begin{cases} 0, & \text{if } t < T \text{ and } a_t \text{ is successful} \\ \sum_{i=0}^{T} \delta_{i+1}^a \delta_{i+1}^b \delta_{i+1}^c, & \text{if } t = T \text{ and } a_t \text{ is successful} \\ \delta_{t+1}^a, & \text{otherwise} \end{cases} \tag{4}$$

where T is the number of iterations of a training episode and where the rewards δ_{i+1}^a, δ_{i+1}^b, and δ_{i+1}^c are defined as follows:

- An Action a_t may lead to a successful or unsuccessful placement. We then define the Acceptance Reward value due to action a_t as

$$\delta_{t+1}^a = \begin{cases} 100, & \text{if } a_t \text{ is successful,} \\ -100, & \text{otherwise.} \end{cases} \tag{5}$$

- The Resource Consumption Reward value for the placement of VNF v via action a_t is defined by

$$\delta_{t+1}^c = \begin{cases} \frac{req_{(v-1,v)}^{bw}}{req_{(v-1,v)}^{bw}|P|} = \frac{1}{|P|}, & \text{if } |P| > 0, \\ 1, & \text{otherwise.} \end{cases} \tag{6}$$

  where P is the path used to place VL $(v-1, v)$. Note that a maximum $\delta_{t+1}^c = 1$ is given when $|P| = 0$, that is, when VNFs $v-1$ and v are placed on the same server.
- The Load Balancing Reward value for the placement of VNF v via a_t, in which parameters $M_{a_t}^{cpu}$ and $M_{a_t}^{ram}$ correspond to initial or maximal amount of CPU and RAM respectively on node a_t.

$$\delta_{t+1}^b = \frac{cap_{a_t}^{cpu}}{M_{a_t}^{cpu}} + \frac{cap_{a_t}^{ram}}{M_{a_t}^{ram}}. \tag{7}$$

B. Adaptation of DRL and Introduction of a Heuristic Function

1) Proposed Deep Reinforcement Learning Algorithm : As in [14], we use a single thread version of the A3C Algorithm introduced in [31]. This algorithm relies on two DNNs that are trained in parallel: i) the Actor Network with the set of parameters θ, which is used to generate the policy π_θ at each time step, and ii) the Critic Network with the set of parameters θ_v which generates an estimate $\nu_{\theta_v}^{\pi_\theta}(\sigma_t)$ for the State-value function defined by

$$\nu_\pi(t|\sigma) = \mathbb{E}_\pi \left[\sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1} | \sigma_t = \sigma \right],$$

for some discount parameter γ.

As depicted in Figure 3 both Actor and Critic Networks have almost identical structure. As in [7], we use the GCN formulation proposed by [32] to automatically extract advanced characteristics of the PSN. The characteristics produced by the GCN represent semantics of the PSN topology by encoding and accumulating characteristics of neighbour nodes in the PSN graph. The size of the neighbourhood is defined by the order-index parameter K. As in [7], we consider in the following $K = 3$ and perform automatic extraction of 60 characteristics per PSN node. The NSPR state characteristics are separately transmitted to a fully connected layer with 4 units. The characteristics extracted by both layers and the GCN layer are combined into a single column vector of size $60|N| + 4$ and passed through a fully connected layer with $|N|$ units.

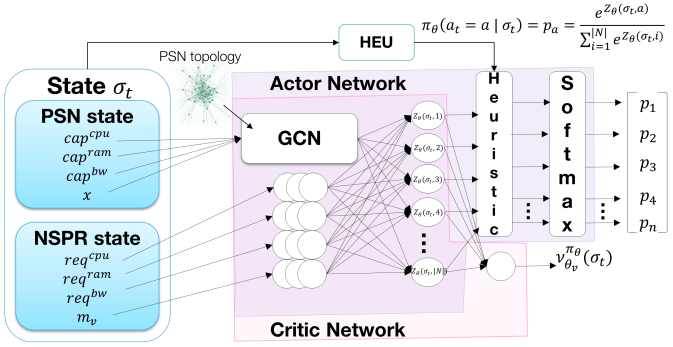


Fig. 3. Reference framework for the proposed learning algorithms.

In the Critic Network, the outputs are forwarded to a single neuron, which is used to calculate the state-value function estimation $\nu_{\theta_v}^{\pi_\theta}(\sigma_t)$. In the Actor Network, the outputs represent the values of the function Z_θ introduced in Section IV-A. These values are injected into a Softmax layer that transforms them into a Softmax distribution that corresponds to the policy π_θ.

During the training phase, at each time step t, the A3C algorithm uses the Actor Network to calculate the policy $\pi_\theta(.|\sigma_t)$. An action a_t is sampled using the policy and performed on the environment. The Critic Network is used to calculate the state-value function approximation $\nu_{\theta_v}^{\pi_\theta}(\sigma_t)$. The learning agent receives then the reward r_{t+1} and next state σ_{t+1} from the environment and the placement process continues until a terminal state is reached, that is, until the Actor Network

returns an unsuccessful action or until the current NSPR is fully placed. At the end of the training episode, the A3C algorithm updates parameters θ and θ_v by using the same rules as in [14].

2) Introduction of a Heuristic Function: To guide the learning process, we use as in [14] the placement heuristic introduced in [29] (see the pseudo code of the heuristic in Algorithm 1). This yields the HA-DRL algorithm. More precisely, from the reference framework shown in Figure 3, we propose to include in the Actor Network the Heuristic layer that calculates a Heuristic Function $H : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ based on external information provided by the heuristic method, referred as HEU.

Let Z_θ be the function computed by the fully connected layer of the Actor Network that maps each state and action to a real value which is after converted by the Softmax layer into the selection probability of the respective action (see Section IV-A). Let $\bar{a}_t = \text{argmax}_{a \in \mathcal{A}} Z_\theta(\sigma_t, a)$ be the action with the highest Z_θ value for state σ_t. Let $a_t^* = HEU(\sigma_t)$ be the action derived by the HEU method at time step t and the preferred action to be chosen. $H(\sigma_t, a_t^*)$ is shaped to allow the value of $Z_\theta(\sigma_t, a_t^*)$ to become closer to the value of $Z_\theta(\sigma_t, \bar{a}_t)$. The aim is to turn a_t^* into one of the likeliest actions to be chosen by the policy.

The Heuristic Function is precisely formulated as

$$H(\sigma_t, a_t) = \begin{cases} Z_\theta(\sigma_t, \bar{a}_t) - Z_\theta(\sigma_t, a_t) + \eta, & \text{if } a_t = a_t^*, \\ 0, & \text{otherwise,} \end{cases} \quad (8)$$

where η parameter is a small real number which can be set so that $H(\sigma_t, a_t^*)$ is always greater or equal to η. If η is greater than 0, there will always be a small increase in the value of $Z(\sigma_t, a_t^*)$ and, therefore, in the probability of choosing the server a_t^* to place the corresponding VNF.

During the training process the Heuristic layer calculates $H(\sigma_t, .)$ and updates the $Z_\theta(\sigma_t, .)$ values by using the following equation:

$$Z_\theta(\sigma_t, .) = Z_\theta(\sigma_t, .) + \xi H(\sigma_t, .)^\beta \quad (9)$$

The Softmax layer then computes the policy using the modified Z_θ. Note that the action returned by a_t^* will have a higher probability to be chosen, but it does not mean that the action a_t^* will be systematically chosen by the DRL agent. The ξ and β parameters are used to control how much HEU influence the policy (i.e., how much we increase the probability of the DRL agent choosing action a_t^*.)

It is worth noting that the main assumption here is that since the HEU method generally produces good placement actions (i.e., actions that bring a high reward), when we use it to control the DRL agent, the DRL agent learns to increase the probability of taking good actions. That is, by taking the actions chosen by the HEU, the DRL agent gets good rewards and consequently learns good behavior.

C. Implementation Remarks

All resource-related characteristics are normalized to be in $[0, 1]$. This is done by dividing cap^j and req^j, $j \in \{$cpu, ram,bw$\}$, by $\max_{n \in N} M_n^j$. With regard to the DNNs, we have implemented the Actor and Critic as two independent Neural Networks. Each neuron has a bias assigned. We have used the hyperbolic tangent (tanh) activation for non-output layers of the Actor Network and Rectified Linear Unit (ReLU) activation for all layers of the Critic Network. We have normalized positive global rewards to be in $[0, 10]$. During the training phase, we have considered the policy as a Categorical distribution and used it to sample the actions randomly.

V. Implementation and Evaluation Results

A. Implementation Details & Simulator Settings

1) Experimental setting: We developed a simulator in Python containing: i) the elements of the Network Slice Placement Optimization problem (i.e., PSN and NSPR); ii) the DRL and HA-DRL algorithms. The DRL algorithm proposed here is based on the A3C algorithm which has shown good results in previous comparative studies for VNE [7] and other problems [31]. It represents a good basis in the comparisons with the HA-DRL algorithm performed in this study. We used the PyTorch framework to implement the DNNs. Experiments were run in a 2x6 cores @2.95Ghz 96GB machine.

2) Physical Substrate Network Settings: We consider a PSN that could reflect the infrastructure of an operator as discussed in [28]. In this network, three types of DCs are introduced as in Section III. Each CDC is connected to three EDCs which are distant of 100 km. CDCs are interconnected and connected to one CCP that is 300 km away.

We consider 15 EDCs each one with 4 servers, 5 CDCs each with 10 servers and 1 CCP with 16 servers. The CPU and RAM capacities of each server are 50 and 300 units, respectively. A bandwidth capacity of 100 Gbps is given to intra-data center links inside CDCs and CCP, 10Gbps being the bandwidth for intra-data center links inside EDCs. Transport links connected to EDCs have 10Gpbs of bandwidth capacity. Transport links between CDCs have 100Gpbs of bandwidth capacity as well for the ones between CDCs and the CCP.

3) Network Slice Placement Requests Settings : We consider NSPRs to have the Enhanced Mobile Broadband (eMBB) setting described in [29]. Each NSPR is composed of 5 or 10 VNFs (see Section V-B2). Each VNF requires 25 units of CPU and 150 units of RAM. Each VL requires 2 Gbps of bandwidth.

B. Algorithms & Experimental Setup

1) Training Process & Hyper-parameters: We consider an online learning scenario, that is, training is not separated for execution. The whole training process has a maximum duration of 6 hours for the considered algorithms. We perform seven independent runs of each algorithm to assess their average performance in terms of the metrics introduced below (see Section V-C).

After having performed Hyper-parameter search (see more details in [14]), we set the learning rates for the Actor and Critic networks of DRL and HA-DRL algorithms to $\alpha = 5 \times 10^{-5}$ and $\alpha' = 1.25 \times 10^{-3}$, respectively.

We program four versions of HA-DRL agents, each with a different value for the β parameter of the heuristic function formulation (see Section IV-B2). We set in addition the parameters $\xi = 1$ and $\eta = 0$.

2) Network load calculation: Network loads are calculated using CPU resource but the analysis could easily be applied to RAM; we use the network load model introduced in Section V-B2. We consider two NSPR classes: i) a Volatile class and ii) a Long term class.

The differences between the two classes are related to their resource requirements and their lifespans as Volatile requests have 5 VNFs and a life-span of 20 simulation time units and Long-term requests have 10 VNFs and a life span of 500 simulation time units.

3) Network load change scenarios: We consider that the network runs in a standard regime under a network load being equal to 40% (i.e., $\rho = 0.4$) and that the NSPRs of each class generate half of the total load. We set slice lifespan parameters $1/\mu^1 = 20$ and $1/\mu^2 = 500$ time units, respectively. The arrival rate parameter λ varies according to the tested scenario. In each experiment, the learning agent is trained during approximately 4 hours for this network load regime. Then a stair-stepped network load change occurs. We simulated eight different network load change levels. Each network load change level is characterized by the addition of a certain amount of extra network load ranging from 10% to 80% (causing system overload).

C. Evaluation Metrics

To characterize the performance of the placement algorithms, we consider two performance metrics:

1) **Average execution time:** the average execution time in seconds required to place 1 NSPR. This metric is calculated based on 100 NSPR placements;
2) **Acceptance Ratio per Training phase (TAR):** this metric represents the Acceptance Ratio obtained in each training phase, i.e., each part of the training process, corresponding to 500 NSPR arrivals or 500 episodes. It is calculated as follows: $\frac{\#\text{accepted NSPRs}}{500}$.

This second metric allows us to better observe the evolution of algorithm performance over time since it measures algorithm performance in independent parts (phases) of the training process without accumulating the performance of previous training phases. For a more comprehensive discussion about the convergence times and training aspects of this algorithms, the interested reader can consult previous work [14].

Based on this metric, we identify three other important metrics used in our results discussion:

1) **Rupture TAR:** it is the TAR obtained in the training phase where the network load change occurs, i.e., the rupture phase;
2) **Last TAR:** it is the TAR obtained in the training phase that is prior to the rupture phase;
3) **Average TAR:** it is the average of the TARs obtained in the 30 phases preceding the rupture phase;
4) **TAR standard deviation:** it is the standard deviation of the TARs obtained in the 30 phases preceding the rupture phase;

D. Execution Time Evaluation

Figure 4a and 4b present the average execution time of the HEU, DRL and HA-DRL algorithms as a function of the number of VNFs in the NSPR and the number of servers in the PSN, respectively (see Section V-C for details on the metric calculation). In both evaluations, we start by the PSN and NSPR settings described in Sections V-A2 and V-A3, respectively, and generate new settings by increasing either the number of VNFs per NSPR or the number of servers in the PSN. The evaluation results confirm our expectations by showing that the average execution times increase faster for heuristics than for a pure DRL approach. However, both HEU and DRL strategies have low execution times (less than 0.6s in the largest scenarios).

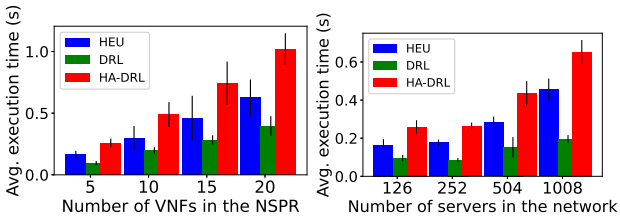


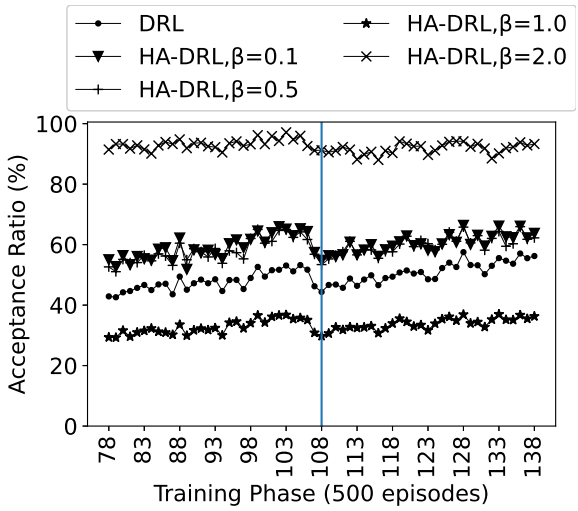
Fig. 4. Average execution time evaluation.

The number of VNFs per NSPR has more impact on the average execution times of HEU and DRL algorithms than the number of servers on the PSN. The average execution time of HEU algorithm is more impacted than DRL by the number of servers in the PSN. The HA-DRL algorithm depends on a sequential execution of DRL and HEU. Therefore, the average execution time of HA-DRL is approximately equal to the sum of the execution times of HEU and DRL. Since DRL and HEU have small execution times, the average execution times of HA-DRL are also small (less than 1.0s for the largest NSPR setting and about 0.6s for the largest PSN setting).

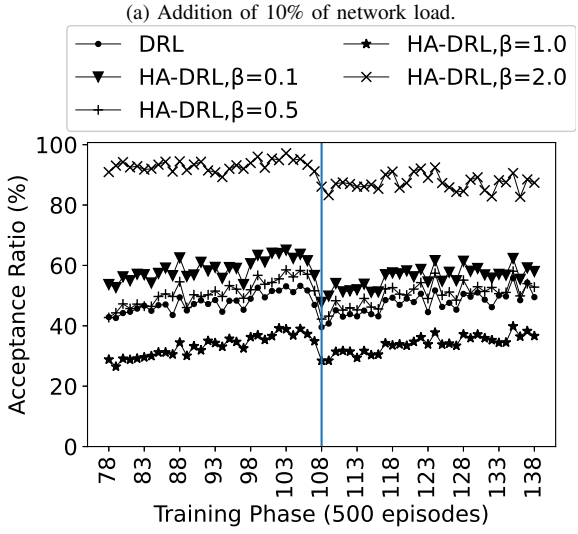
E. Evaluation of the impact of network load change

Figure 5, 6 and 7 capture the impact of different network load change levels on the TARs obtained by the different evaluated algorithms. The rupture phase is identified by a blue vertical line in the various figures. The performance of the algorithms seems to drop before the blue line. This impression is caused by expected performance variations since the training process is not yet completed and the performances of the algorithms are not yet stable after only a few hours of training. With the reduced training time of 6 hours, the only algorithm that has near optimal performance after 108 training phases is HA-DRL, with $\beta = 2.0$. This is due to the fact that the strong influence of the Heuristic Function helps the algorithm to become stable more quickly as discussed in [14] and [15].

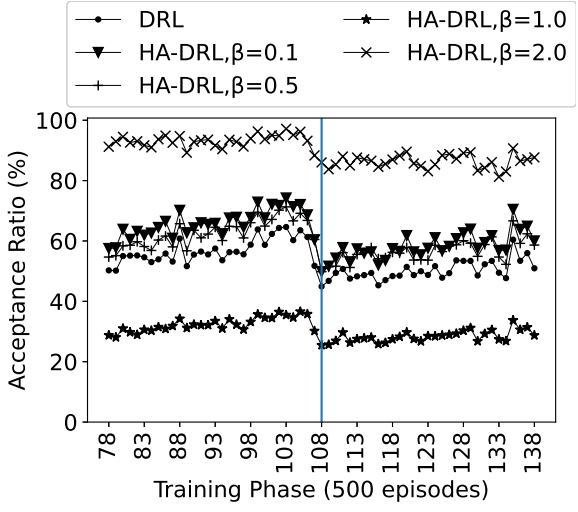
We can also observe by the shape of the different curves in Fig. 5, 6, and 7 that, as expected, all the algorithms have some variability in their performance during the training phases. In addition, these figures show that the performance of all the algorithms is affected at various levels by the network load change and that, generally speaking, the higher the amount of extra network load added, the lower is the TAR after the



(a) Addition of 10% of network load.

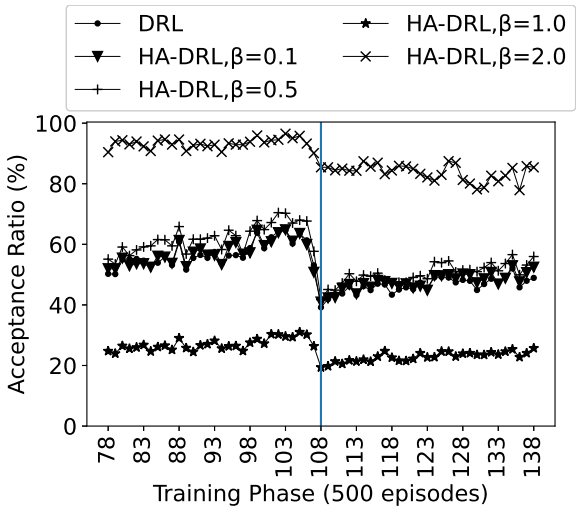


(b) Addition of 20% of network load.

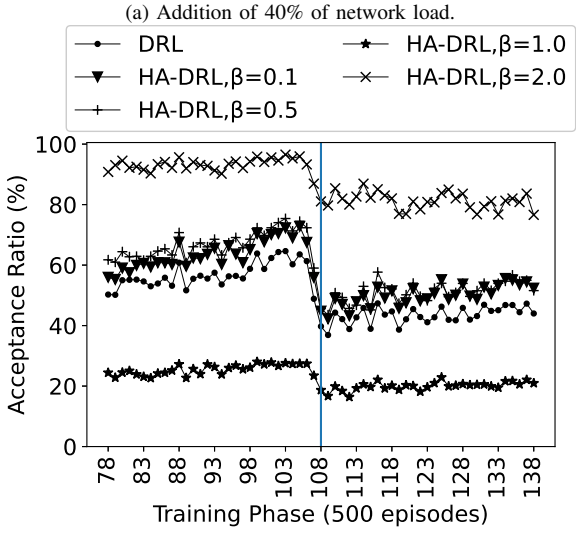


(c) Addition of 30% of network load.

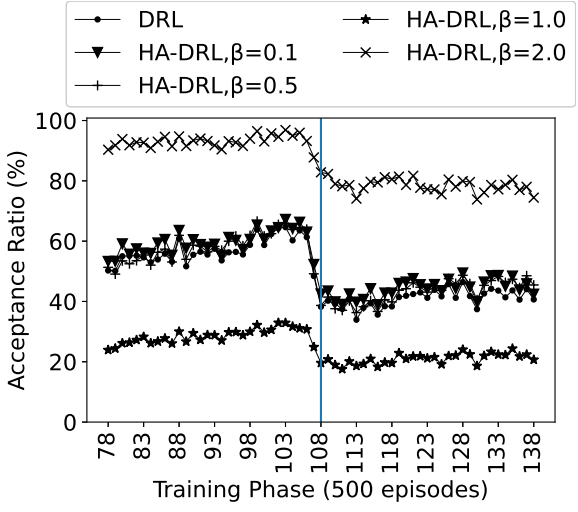
Fig. 5. Evaluation of impact of network load disruption on TAR: under-loaded scenarios



(a) Addition of 40% of network load.

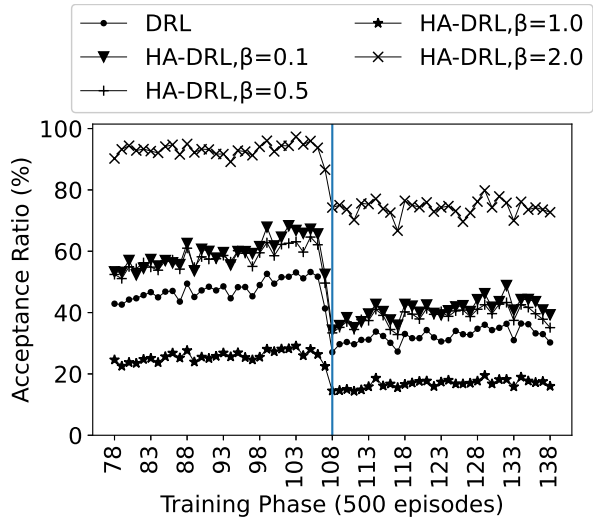


(b) Addition of 50% of network load.

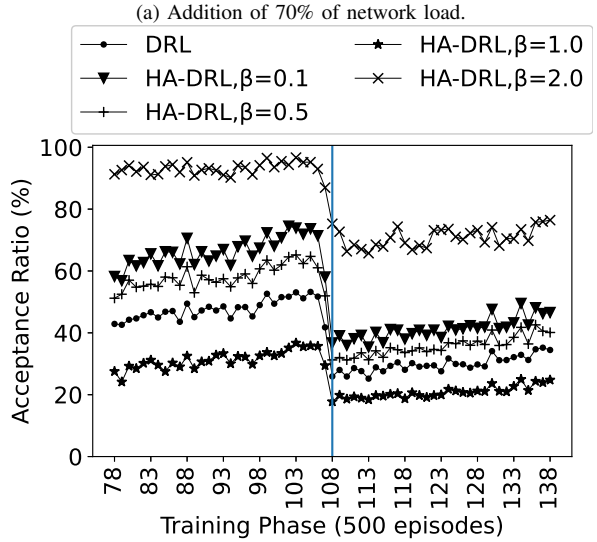


(c) Addition of 60% of network load.

Fig. 6. Evaluation of impact of network load disruption on TAR: critical scenarios



(a) Addition of 70% of network load.



(b) Addition of 80% of network load.

Fig. 7. Evaluation of impact of network load disruption on TAR: overloaded scenarios

change. Finally, we can also see that the only algorithm to keep a near optimal performance even in overloaded scenarios shown in Fig. 7 is HA-DRL, with $\beta = 2.0$.

Tables III, IV,V, VI, and VII present other performance metrics related to the various evaluated algorithms. The columns "Rupture TAR - Avg. TAR" and "Rupture TAR - Last TAR" indicate how much the performance of the algorithms drops in the rupture phase when compared with the Average TAR and Last TAR, respectively. The TAR Standard Deviation column indicates the TAR Standard Deviation metric described in Section V-C.

Those tables confirm that in general the performance gaps, i.e., the gaps between the Rupture TAR and Average or Last TAR, grow with the level of disruption for all algorithms. For instance, in the disruption level "+10", the performance gaps are never higher than 5%. But, in the change level "+80" the performance gaps are never lower than 11%.

In all the evaluated cases, the difference between the Rupture TAR and the Average TAR is higher than the TAR standard deviation. For instance, for the DRL algorithm, in network load disruption level of +50%, rupture TAR is 17% lower than Average TAR which is 3.94 times the TAR standard deviation.

TABLE III
DRL ALGORITHM RESULTS

Network Load Disruption Level (%)	Rupture TAR - Avg. TAR (%)	Rupture TAR - Last TAR (%)	TAR Standard Deviation (%)
+10	-3.37	-1.89	3.10
+20	-8.19	-7.37	3.09
+30	-11.89	-6.83	4.17
+40	-17.68	-13.8	4.12
+50	-17.00	-9.11	4.32
+60	-18.50	-10.20	4.35
+70	-20.46	-14.26	3.30
+80	-21.65	-15.86	3.27

TABLE IV
HA-DRL, $\beta = 0.1$ ALGORITHM RESULTS

Network Load Disruption Level (%)	Rupture TAR - Avg. TAR (%)	Rupture TAR - Last TAR (%)	TAR Standard Deviation (%)
+10	-4.13	-2.60	4.07
+20	-11.02	-8.91	3.51
+30	-16.00	-10.54	4.50
+40	-16.28	-9.83	4.13
+50	-18.66	-11.14	5.05
+60	-17.02	-9.80	3.99
+70	-25.13	-18.20	4.93
+80	-29.41	-21.31	4.85

The algorithm with the lower performance gaps is HA-DRL with $\beta = 1.0$ as we can see in columns "Rupture TAR - Avg. TAR" and "Rupture TAR - Last TAR" of Table VI. We can state that this algorithm has significantly better robustness than all the others as performance gaps are significantly lower. However, HA-DRL with $\beta = 1.0$ has the worst TAR performance as shown in Fig. 5, 6 and 7, which reduces its applicability.

HA-DRL with $\beta = 2.0$ has the second better robustness and DRL the third as we can see on "Rupture TAR - Avg. TAR" and "Rupture TAR - Last TAR" columns of Tables VII and III, respectively. Even if the usage of the Heuristic Function has helped HA-DRL, with $\beta \in \{0.1, 0.5\}$ to achieve significantly better TARs than DRL, the influence of the Heuristic Function in these algorithms was not sufficient to improve the robustness of the DRL algorithm against unpredictable network load disruptions (see Tables IV and V, respectively).

We can observe, however, that HA-DRL with $\beta = 2.0$ has better robustness against unpredictable network load changes than DRL as the performance gaps obtained with HA-DRL with $\beta = 2.0$ are significantly lower than the ones obtained with DRL as can be observed in columns "Rupture TAR - Avg. TAR" and "Rupture TAR - Last TAR" of Tables VII and III, respectively. These results confirm that HA-DRL with $\beta = 2.0$ is the algorithm among those evaluated that is the most adapted to be used in practice. Indeed, the algorithm presents not only the better TAR results and quick convergence but also robust performance.

TABLE V
HA-DRL, $\beta = 0.5$ ALGORITHM RESULTS

Network Load Disruption Level (%)	Rupture TAR - Avg. TAR (%)	Rupture TAR - Last TAR (%)	TAR Standard Deviation (%)
+10	-4.55	-3.43	3.95
+20	-8.80	-9.37	4.21
+30	-12.78	-10.66	4.59
+40	-20.33	-15.94	4.61
+50	-21.24	-13.43	4.56
+60	-19.46	-10.46	5.08
+70	-24.28	-16.26	3.75
+80	-26.78	-20.71	3.88

TABLE VI
HA-DRL, $\beta = 1.0$ ALGORITHM RESULTS

Network Load Disruption Level (%)	Rupture TAR - Avg. TAR (%)	Rupture TAR - Last TAR (%)	TAR Standard Deviation (%)
+10	-2.96	-1.11	2.37
+20	-4.94	-6.49	3.50
+30	-6.93	-4.71	2.37
+40	-7.67	-7.00	1.97
+50	-6.80	-4.77	1.72
+60	-8.95	-5.29	2.45
+70	-11.25	-8.00	1.73
+80	-13.62	-11.69	2.89

TABLE VII
HA-DRL, $\beta = 2.0$ ALGORITHM RESULTS

Network Load Disruption Level (%)	Rupture TAR - Avg. TAR (%)	Rupture TAR - Last TAR (%)	TAR Standard Deviation (%)
+10	-2.04	0.09	2.37
+20	-7.01	-5.09	3.50
+30	-7.15	-2.31	2.37
+40	-7.90	-4.69	1.97
+50	-12.13	-5.86	1.72
+60	-10.24	-4.94	2.45
+70	-18.83	-12.34	1.73
+80	-17.79	-11.69	2.89

VI. CONCLUSION

We have introduced in this paper two DRL-based algorithms and evaluated their performance in a non-stationary network load scenario with unpredictable changes. In line with the conclusions of [14], [15], the numerical experiments performed in this paper show that coupling DRL and heuristic functions yields good and stable results even under non stationary load conditions. Therefore, we believe that such an approach is relevant in real networks that are subject to unpredictable network load changes.

As part of our future work, we plan to explore distribution and parallel computing techniques to solve the considered multi-objective optimization problem using multi-agent or federated learning approaches to address slice placement in heterogeneous networks, mainly when the network is decomposed into several segments or technical domains, where the network abstraction introduced in this paper is no more valid. Indeed, each segment should have its own abstractions and data. It is then necessary to share information between the segments to take a global decision. Instead of exchanging complete network states, segments would exchanging minimal information obtained via heuristics.

ACKNOWLEDGMENT

This work has been performed in the framework of 5GPPP MON-B5G project (www.monb5g.eu). The experiments were conducted using Grid'5000, a large scale testbed by Inria and Sorbonne University (www.grid5000.fr).

REFERENCES

[1] 3GPP, "Management and orchestration; 5G Network Resource Model (NRM); Stage 2 and stage 3 (Release 17)," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 28.541, Dec. 2020, version 17.1.0.

[2] ETSI NFV ISG, "Network Functions Virtualisation (NFV); Evolution and Ecosystem; Report on Network Slicing Support, ETSI Standard GR NFV-EVE 012 V3.1.1," ETSI, Tech. Rep., 2017. [Online]. Available: https://www.etsi.org/technologies-clusters/technologies/nfv

[3] J. Gil Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, pp. 518–532, Sep. 2016.

[4] A. Laghrissi and T. Taleb, "A survey on the placement of virtual resources and virtual network functions," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1409–1434, 2nd. Quart., 2019.

[5] J. J. A. Esteves, A. Boubendir, F. Guillemin, and P. Sens, "Location-based data model for optimized network slice placement," in *Proc. 2020 6th IEEE Conf. Netw. Softwarization (NetSoft)*, 2020, pp. 404–412.

[6] E. Amaldi, S. Coniglio, A. M. Koster, and M. Tieves, "On the computational complexity of the virtual network embedding problem," *Electron. Notes Discrete Math.*, vol. 52, pp. 213–220, Jun. 2016.

[7] Z. Yan, J. Ge, Y. Wu, L. Li, and T. Li, "Automatic virtual network embedding: A deep reinforcement learning approach with graph convolutional networks," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1040–1057, Jun. 2020.

[8] M. Dolati, S. B. Hassanpour, M. Ghaderi, and A. Khonsari, "DeepViNE: Virtual network embedding with deep reinforcement learning," in *Proc. IEEE INFOCOM 2019 - IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, 2019, pp. 879–885.

[9] 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, Apr. 2018.

[10] H. Wang, Y. Wu, G. Min, J. Xu, and P. Tang, "Data-driven dynamic resource scheduling for network slicing: A deep reinforcement learning approach," *Inf. Sci.*, vol. 498, pp. 106–116, Sep. 2019.

[11] Y. Xiao, Q. Zhang, F. Liu, J. Wang, M. Zhao, Z. Zhang, and J. Zhang, "NFVdeep: Adaptive online service function chain deployment with deep reinforcement learning," in *Proc. 2019 IEEE/ACM 27th Int. Symp. Qual. Service (IWQoS)*, 2019, pp. 1–10.

[12] P. T. A. Quang, A. Bradai, K. D. Singh, and Y. Hadjadj-Aoul, "Multi-domain non-cooperative VNF-FG embedding: A deep reinforcement learning approach," in *Proc. IEEE INFOCOM 2019 - IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, 2019, pp. 886–891.

[13] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. Cambridge, MA, USA: MIT press, 2015.

[14] J. J. Alves Esteves, A. Boubendir, F. Guillemin, and P. Sens, "A heuristically assisted deep reinforcement learning approach for network slice placement," *arXiv preprint arXiv:2105.06741*, 2021.

[15] J. J. A. Esteves, A. Boubendir, F. Guillemin, and P. Sens, "Drl-based slice placement under non-stationary conditions," *Submitted to IEEE 17th International Conference on Network and Service Management (CNSM)*, 2021.

[16] P. T. A. Quang, Y. Hadjadj-Aoul, and A. Outtagarts, "A deep reinforcement learning approach for vnf forwarding graph embedding," *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 4, pp. 1318–1331, Dec. 2019.

[17] A. Rkhami, Y. Hadjadj-Aoul, and A. Outtagarts, "Learn to improve: A novel deep reinforcement learning approach for beyond 5G network slicing," in *Proc. 2021 IEEE 18th Annu. Consum. Commun. Netw. Conf. (CCNC)*, 2021, pp. 1–6.

[18] J. Pei, P. Hong, M. Pan, J. Liu, and J. Zhou, "Optimal vnf placement via deep reinforcement learning in sdn/nfv-enabled networks," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 2, pp. 263–278, Feb. 2020.

[19] D. Bertsimas and A. Thiele, "Robust and data-driven optimization: modern decision making under uncertainty," in *Models, methods, and applications for innovative decision making*. INFORMS, 2006, pp. 95–122.
[20] M. Shafique, M. Naseer, T. Theocharides, C. Kyrkou, O. Mutlu, L. Orosa, and J. Choi, "Robust machine learning systems: Challenges, current trends, perspectives, and the road ahead," *IEEE Design & Test*, vol. 37, no. 2, pp. 30–57, Apr. 2020.
[21] R. R. O. Al-Nima, T. Han, S. A. M. Al-Sumaidaee, T. Chen, and W. L. Woo, "Robustness and performance of deep reinforcement learning," *Applied Soft Comput.*, vol. 105, p. 107295, Jul. 2021.
[22] A. Marotta, E. Zola, F. D'Andreagiovanni, and A. Kassler, "A fast robust optimization-based heuristic for the deployment of green virtual network functions," *J. Netw. Comput. Applications*, vol. 95, pp. 42–53, Jul. 2017.
[23] A. Marotta, F. D'andreagiovanni, A. Kassler, and E. Zola, "On the energy cost of robustness for green virtual network function placement in 5g virtualized infrastructures," *Comput. Netw.*, vol. 125, pp. 64–75, Apr. 2017.
[24] V. S. Reddy, A. Baumgartner, and T. Bauschert, "Robust embedding of vnf/service chains with delay bounds," in *2016 IEEE Conf. Netw. Funct. Virtualization Softw. Defined Netw. (NFV-SDN)*. IEEE, 2016, pp. 93–99.
[25] A. Baumgartner, T. Bauschert, A. A. Blzarour, and V. S. Reddy, "Network slice embedding under traffic uncertainties — a light robust approach," in *2017 13th Int. Conf on Netw. Service Manag. (CNSM)*. IEEE, 2017, pp. 1–5.
[26] P. Sun, J. Lan, J. Li, Z. Guo, and Y. Hu, "Combining deep reinforcement learning with graph neural networks for optimal vnf placement," *IEEE Commun. Letters*, vol. 25, no. 1, pp. 176–180, Jan. 2021.
[27] F. Slim, F. Guillemin, and Y. Hadjadj-Aoul, "CLOSE: A costless service offloading strategy for distributed edge cloud," in *Proc. 2018 15th IEEE Annu. Cons. Commun. Netw. Conf. (CCNC)*, 2018, pp. 1–6.
[28] F. Slim, F. Guillemin, A. Gravey, and Y. Hadjadj-Aoul, "Towards a dynamic adaptive placement of virtual network functions under ONAP," in *Proc. 2017 IEEE Conf. on Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, 2017, pp. 210–215.
[29] J. J. Alves Esteves, A. Boubendir, F. Guillemin, and P. Sens, "Heuristic for edge-enabled network slicing optimization using the "power of two choices"," in *Proc. 2020 IEEE 16th Int. Conf. Netw. Service Manag. (CNSM)*, 2020, pp. 1–9.
[30] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 10, pp. 1094–1104, Oct. 2001.
[31] 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 *Int. Conf. Mach. Learn.* PMLR, 2016, pp. 1928–1937.
[32] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. 5th Int. Conf. Learn. Representations (ICLR)*, 2017, pp. 1–14.

José Jurandir Alves Esteves graduated from the University of Clermont Auvergne in 2017 and from the Federal University of Minas Gerais in 2019 obtaining two engineering degrees and a master's degree in computer science from the University of Clermont Auvergne. He is doing his PhD between Orange Labs and the Computer Science Laboratory of Paris 6 (LIP6), Sorbonne University. His research is related to automation and optimization models and algorithms for network slice orchestration.

Amina Boubendir is a researcher and project manager at Orange Labs in France. Her research is in the area of design, management and softwarization of networks and services. Amina received a Master degree in Network Design and Architecture from Télécom Paris in 2013, and a PhD in Networking and Computer Science from Télécom Paris in 2016. She is a member of the Orange Expert community on "Networks of the Future".

Fabrice Guillemin graduated from Ecole Polytechnique in 1984 and from Telecom Paris in 1989. He received the PhD degree from the University of Rennes in 1992. He defended his "habilitation" thesis in 1999 at the University Pierre et Marie Curie (LIP6), Paris. Since 1989, he has been with Orange Labs (former CNET and France Telecom R&D). He is currently leading a project on the evolution of network control. He is a member of the Orange Expert community on "Networks of the Future".

Pierre Sens received his Ph.D. in Computer Science in 1994, and the "Habilitation à diriger des recherches" in 2000 from Paris 6 University (UPMC), France. Currently, he is a full Professor at Sorbonne Université (ex-UPMC). His research interests include distributed systems and algorithms, large scale data storage, fault tolerance, and cloud computing. He is leading Delys a joint research team between LIP6 and Inria. He was member of the Program Committee of major conferences in the areas of distributed systems and parallelism (ICDCS, IPDPS, OPODIS, ICPP, Europar, SRDS, DISC. . .) and has served as general chair of SBAC-PAD and EDCC. Overall, he has published over 150 papers in international journals and conferences and has acted for advisor of 25 Ph.D. thesis.