This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3148703, IEEE Access

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Admission Control and Virtual Network Embedding in 5G Networks: a Deep Reinforcement-Learning approach

SEBASTIAN TROIA, ANDRES F. R. VANEGAS, LIGIA M. M. ZORELLO, and GUIDO MAIER
Department of Electronic, Information and Bioengineering (DEIB), Politecnico di Milano, Milan, Italy

Corresponding author: Sebastian Troia (e-mail: sebastian.troia@ polimi.it).

ABSTRACT Fifth-generation (5G) networks are already available in major urban areas and are expected to bring a major transformation to citizens' lives. 5G services, such as enhanced mobile broadband (eMBB), ultra-reliable low latency communications (URLLC), and massive machine-type communications (mMTC), require a network infrastructure capable of supporting stringent requirements in terms of latency and bandwidth demands; as such, it must be highly dynamic and flexible. Network slicing is a key enabler technology that can provide dynamic and flexible characteristics to 5G network architecture. A network slice (NS) can be defined as a partition of network and IT resources, that is, network links and nodes capacity dedicated to a specific set of service demands. As a result, different NSs can coexist over the same physical infrastructure network and can be used to dynamically and flexibly deploy the aforementioned 5G services. However, to efficiently implement NSs with different requirements, communication service providers (CSPs) that own the physical infrastructure network must adopt sophisticated techniques for admission control and resource allocation of NSs. In this paper, we present a novel framework for admission control and resource allocation of 5G NSs in metro-core networks. Specifically, our framework is based on a deep reinforcement learning (DRL) algorithm called Advantage Actor Critic (A2C), which performs admission control, i.e. it is capable of learning which slice to admit based on the availability of the physical network resources. Then, given the diversity of requirements for each 5G service, we propose different resource allocation algorithms based on integer linear programming (ILP) and heuristics to treat each service accordingly. The results show that our proposed framework can increase the number of admitted NSs with respect to the case in which the admission control is disabled by improving the resource allocation performance.

INDEX TERMS Network slicing, Network Function Virtualization (NFV), 5G, Deep Reinforcement Learning

I. INTRODUCTION

Emerging 5G technology is now widely available in major urban areas, and coverage is expected to reach less populated areas in the coming years [1].

5G services, such as enhanced mobile broadband (eMBB), ultra-reliable low latency communications (URLLC), and massive machine-type communications (mMTC), are defined by the International Telecommunication Union (ITU) [2] and will soon be available to the majority of citizens.

In addition, 5G has already reached relevant industrial scenarios, thanks to the introduction of new use cases enabled by 5G connectivity that have improved the productivity and performance of the production chain, for example, Industrial IoT (IIoT).

5G leverages the benefits of network function virtualization (NFV) to accommodate flexibility in providing carrier-grade differentiated services. The notion of NFV focuses on the concept of a software-based representation of both hardware and software resources by considering data and/or control-plane functions. In other words, NFV is the paradigm of moving network functions, such as routing, firewall, and NAT, from dedicated hardware appliances to software-based applications running on commercial off-the-shelf equipment [3]. These virtual network functions (VNFs) provide many benefits to communication service providers (CSPs) that own the physical infrastructure network. They enable openness of platforms, scalability and flexibility, shorter development cycles, and reduced capital expenditure (CapEx) and operating expenditure (OpEx) [4].

NFV is the main foundation of network slicing, which ensures isolation and multi-tenancy support on a common physical network infrastructure by enabling logical and physical separation of network resources. Specifically, a network slice (NS) is defined as a partition of network and IT resources, that is, network links and nodes capacity dedicated to a specific set of service demands. As such, different NSs can coexist over the same physical substrate network (SN) and can be used to dynamically and flexibly interconnect VNFs by providing different types of services, such as real-time video streaming and enterprise services. To such extent, NFV opens up the implementation and management of NSs not only to CSPs, which are also infrastructure network providers (InPs), but also to third-party service providers, such as Network Slice Providers (NSPs), which rely on one or more InPs to sell 5G services to end users.

Although the adoption of NFV brings revolutionary benefits in terms of scale and agility, it also brings a new level of complexity. Virtualization breaks traditional networking into dynamic components and layers that have to work in unison and can change at any given time [5]. For instance, a virtualized firewall can be subject to continuous updates by NSPs. To efficiently implement NSs over SNs, NSPs must deploy a software-based embedding system (ES) comprising a set of sophisticated techniques for admission control and resource allocation of NSs of different types, such as eMBB, URLLC, and mMTC.

The problem of how to allocate physical resources to virtual resources is called virtual network embedding problem (VNEP). In most real-world scenarios, the VNEP needs to be addressed as an online problem (online VNEP). That is, we do not know how many and which types of Network Slice Requests (NSRs) will come to the ES, as such, they arrive dynamically and remain in the SN for an arbitrary period of time [6]. To be realistic, the ES must handle the NSRs as they arrive through an admission control (AC) algorithm, rather than attending a set of NSRs at once (offline VNEP).

The NSP may decide to admit NSRs deemed to have the best chance of meeting the predefined requirements. For example, the URLLC and eMBB are two dominant types of service of the emerging 5G network. Latency and reliability are major concerns for URLLC NSRs (0.25-0.30 ms/packet [7]), while eMBB NSRs request for the maximum data rates (Gbps). The trade-off among latency and reliability between eMBB and URLLC services, heads to a challenging scheduling dilemma [8] [9].

Slice admission is also dictated by the available resources in the network resource pool, and the AC algorithm must consider the available resources in the SN and manage them in order to accommodate as many NSRs as possible.

Different solutions to this problem have been addressed by several research works by employing various techniques, such as Markov chains [10], big data analytics [11], queuing theory [12], etc. (see Section II).

However, these studies do not differentiate the slices embedding according to the 5G services they carry, i.e. eMBB, URLLC, and mMTC.

This paper proposes a novel ES framework to solve the online VNEP of 5G services in metro-core networks. The AC algorithm is based on deep reinforcement learning (DRL) while the VNE is based on integer linear programming (ILP) and heuristic algorithms.

DRL is a subfield of machine learning that aims to capture the most important features of a dynamic environment by deploying a learning agent (powered by deep learning algorithms) that interacts with it to achieve a goal [13]. For instance, authors in [14] developed AlphaZero, a DRL framework that performs exceptionally in games such as chess, shogi, and Go. In short, this framework includes a set of software agents capable of learning how to win in these games. They generate a series of actions (such as movements of pawns in the chessboard) based on the results produced in previous games. Each time they play, they produce increasingly better results. An interesting aspect of DRL is that it implements software agents capable of learning how to optimize an objective function by interacting with an environment that can assume hundreds of thousands of different states. For this reason, given the complexity of the AC problem, our goal is to implement a DRL algorithm to optimize the admission of NSRs in a 5G metro-core network environment. In particular, we implement a novel algorithm called Advantage Actor Critic (A2C) [15], which combines two types of reinforcement learning algorithms: policy-based and value-based. Policy-based agents directly learn a policy (probability distribution of actions) by mapping input states to output actions. Value-based algorithms learn to select actions based on the predicted value of the input state or action. Moreover, given the different requirements of 5G services, we developed different ILP and heuristic-based algorithms to address the VNE of eMBB, URLLC, and mMTC slices. This work considers the metro-core network architecture proposed by the Metro-Haul European project [16] as an SN. This network defines an NFV infrastructure that comprises metro nodes with IT and TLC equipment, following the multi-access edge computing (MEC) model defined by ETSI to support the instantiation of VNFs. Both ILPs and heuristics aim to optimize the SN resources for each NSR.

The remainder of this paper is organized as follows. Section II presents the related work. Section III presents the proposed ES framework that solves the online VNEP of 5G slices. Section IV presents the performance evaluation of the proposed AC and VNE algorithms. Finally, section V concludes the study.

II. RELATED WORK AND PAPER CONTRIBUTION

This section reviews the research works that have investigated AC and VNE algorithms.

A. ADMISSION CONTROL TECHNIQUES

The authors in [10] presented an analytical model based on a semi-Markov decision process enhanced by an artificial neural network (ANN) to perform AC of NSs on a wireless access network. The objective of the model is to maximize the overall profit of the infrastructure network provider while guaranteeing the service level agreement (SLA) committed to all slices.

In ref. [11], the authors introduced a slice admission strategy based on big data analytics (BDA) predictions. They considered a network architecture comprising three different domains: 1) wireless access network, 2) metro-optical network, and 3) datacenter (cloud) network. The goal is to accept a slice request issued by a customer from the wireless network domain only when it is estimated that no service degradation will occur for both the incoming slice request and the slices already deployed. The BDA prediction algorithm consists of a regression-based framework that makes predictions based on past data.

Han et al. [12] tackled the AC problem by proposing a system based on the queuing theory. The authors cast this problem into a typical wireless access network scenario, where the mobile operator decides to lease infrastructure resources to customers (or tenants). The proposed system consists of a stochastic model that leverages a multi-queuing system (e.g., one queue for each type of slice) to design an AC for on-demand network slices.

Challa et al. [17] mapped the AC problem into a knapsack problem (MKP) with randomized arrivals and slice durations. The goal is to maximize resource monetization, defined as the revenue of the network provider while minimizing the rejection rate to avoid SLA violations.

The authors in [18] proposed an AC model based on RL with the goal of maximizing the profit of the network provider. Specifically, they considered a 5G flexible RAN, where slices of different mobile service providers are virtualized over the same RAN infrastructure. The proposed RL-based algorithm employs an ANN-based stochastic policy network to model the AC agent used to accept or deny slice request to maximize revenue.

In [19], the authors proposed a framework for network slice management in the context of a 5G RAN. It comprises three modules: prediction, AC, and scheduling. The prediction module is responsible for predicting the traffic of a specific slice. The second module performs the AC as a geometric knapsack problem, showing that this problem is NP-hard. Finally, the scheduling module is in charge of meeting the agreed SLAs, and reports back deviations to the prediction module.

In [20], the authors proposed an admission control based on a recurrent neural network (RNN) to improve the overall system performance for the online VNE problem. The admission control serves as a filter for the incoming network slices by preventing the VNE algorithms from spending time on slices that are either infeasible or that cannot be embedded within an acceptable time. Their approach was based on supervised learning, which means that their RNN algorithm is trained offline.

The different techniques proposed in the aforementioned papers are based on accepting/rejecting individual network slices as they arrive. These studies do not distinguish the slices according to the 5G services [21], such as eMBB, URLLC, and mMTC, preventing the diversity of their QoS requirements. Furthermore, most of the proposed techniques focus on allocating radio resources while neglecting the allocation of 5G metro-core network nodes. In this study, we focus on a generic 5G metro-core network and consider standardized 5G services to perform AC based on a novel DRL approach.

B. VIRTUAL NETWORK EMBEDDING TECHNIQUES

In this section, we present an overview of the research work on the VNE techniques. Several research works have proposed different methodologies to solve this problem. ILP formulations, hence, mathematical optimization models, provide optimal solutions. Heuristics algorithms cope with the complexity problems that generally affect ILPs, such as the rapidly-increasing resolution time due to the large number of variables and constraints. Machine learning algorithms provide flexible and autonomous solutions to cope with the dynamic nature of VNE problems.

In [22], the authors proposed an ILP formulation to solve the VNE. The authors implemented a multi-commodity flow formulation to optimize the allocation of a generic slice onto a physical network. Their model strives to minimize physical resource consumption and load balancing, which is accomplished by means of three different objective functions: load balancing plus shortest path (LB+SP), shortest distance path (SDP), and weighted shortest distance path (WSDP).

In [23], the authors proposed a heuristic algorithm for the VNE. The authors evaluated the acceptance ratio and run time of rank algorithms by comparing them with an ILP-based solution. They claim that all ranking techniques achieve high acceptance ratios within short run-times, whereby the best algorithm depends mainly on the slice and the SN.

In [24], the authors provided an ILP formulation by proposing a single objective function to compute the optimal VNE with respect to the revenue-to-cost ratio. They considered computation power, memory, throughput, and latency as the relevant resources for the SN by providing a nearly optimal ILP formalization and implementation. However, the revenue and cost are not the only objectives that an embedding algorithm needs to meet for every type of slice, different types of slices have different requirements, and as they claim in the future work section, heuristics need to be performed and evaluated to solve large problem instances within a short run-time.

The authors in [25] proposed a graph-based model to map network slices to the SN. The mapping process is based on two steps: 1) node mapping, that is, the selection of the substrate nodes as the host for the virtual nodes of the slices;

2) then, link mapping, that is, the procedure of connecting selected host nodes in the physical SN.

In [20], the authors formulate an ILP and a heuristic algorithm for VNE of network slices. They considered network slices requests with a delay requirement and a set of service function chains (SFCs) where each SFC is denoted as a set of VNFs with a capacity requirement. However, this solution does not have different objectives of bandwidth or CPU capacity for the different types of slices; their work aims to minimize the embedding costs while guaranteeing delay constraints.

In [26], the authors proposed a framework based on deep reinforcement learning to allocate virtual radio resources. They considered bandwidth- and delay-constrained slices, whose radio resources are mapped to base stations to maximize the transmission rate and minimize the queuing delays according to the slice requirements. Nevertheless, this work does not consider the slice admission control nor the metro-core network slicing.

In [27], the authors developed an algorithm for VNE based on deep reinforcement learning. They considered virtual network requests that demand node CPU processing and link bandwidth resources to the network with the goal of improving the acceptance ratio and the revenue. Besides enabling dynamic slice embedding, this work does not consider the heterogeneous requirements of the services that may be carried by the slice.

Most part of these studies mapped the arriving generic slices onto the physical SN.

In contrast, in this work, we propose both ILP and heuristic-based algorithms that consider four types of 5G network slices, namely, generic, eMBB, mMTC, and URLLC, according to the International Telecommunication Union (ITU) [21]. We consider their requirements to tailor both the proposed ILP and heuristic-based VNE algorithms. In short, the objective of this work is to jointly perform AC and VNE of NSs in a 5G metro-core network by optimizing resource utilization and maximizing the revenue of the infrastructure network provider.

C. PAPER CONTRIBUTION

In a nutshell, we developed a novel ES framework able to solve the online VENP for 5G services in metro-core networks. The contribution of this work can be summarized as follows:

- We implemented a novel algorithm for the AC of NSs based on DRL. In particular, we tailored the Advantage Actor Critic (A2C) [15] algorithm with the aim of optimizing the slice admission control of different types of 5G services, such as generic, eMBB, mMTC, and URLLC. Furthermore, we took into account the revenue-to-cost ratio after the VNE procedure to design the reward function.
- We included two types of VNE algorithms in our ES that are inspired by the research work in [25]. The first is based on an ILP mathematical formulation with the aim of solving the online VNE for different types of NSR. The second is based on a heuristic with the goal of reducing the computation time of ILP-based algorithms. Compared to the work in [25], we provided the following novelties:
    1) We extended the ILP formulation by modifying the objective functions for eMBB and mMTC NSRs. In particular, our objective functions ensure that the residual physical resources on the SN are maximized. As a result, our ILP formulations provided a fair distribution of available resources throughout the SN in order to prevent some nodes, or links, from being used more than others.
    2) We developed the heuristic algorithms by taking into account specific requirements of the 5G NSRs. As such, we have considered the maximum CPU capacity of the physical nodes as the main feature to embed the eMBB NSRs; then, we have used the minimum bandwidth available on the substrate nodes and the number of hops as the main features for the mMTC and URLLC NSRs respectively.

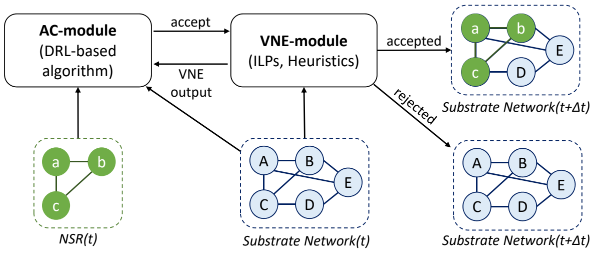

FIGURE 1: Embedding system framework for AC and VNE

In the following sections, we discuss the proposed ES that solves the online VNEP for 5G services on a metro-core network.

III. EMBEDDING SYSTEM FRAMEWORK FOR 5G NETWORK SLICES

Fig. 1 shows the proposed ES framework. It includes two modules: an AC module and a VNE module. The first is based on a DRL algorithm and performs admission control of NSRs. The second is based on different ILP and heuristic-based algorithms and performs resource allocation onto the SN. The proposed ES framework is as follows:

1) At time t, an NSR arrives at the AC module. It decides to accept or reject the NSR based on the output of the VNE module and the status of the SN at the previous time step.
2) If the AC module accepts the NSR, it informs the VNE module such that it can be embedded onto the SN. Otherwise, it is rejected.
3) Once the VNE module receives acceptance from the AC module, it performs the embedding procedure. If the NSR embeds successfully, the VNE module sends positive feedback to the AC module; otherwise, it sends

negative feedback. The latter is important for the AC module because it can learn to reject NSRs that cannot be embedded in advance.

The following sections will discuss the details of the SN, the type of NSRs, the DRL-based AC module, and the VNE module.

A. SUBSTRATE NETWORK (SN)

The SN considered in this work follows the metro-core network architecture proposed by the Metro-Haul European project [16]. The Metro-Haul (MH) network is designed to support network slicing according to the 5G-MEC model; as such, it can support different 5G service requirements. For instance, eMBB requires a wide range of VNFs distributed across the core and edge metro nodes. At the same time, the mMTC needs VNFs deployed at the edge to support a high connection density of online devices such as sensors and other wireless devices. The MH network defines an NFV infrastructure that comprises metro core edge nodes (MCENs) interconnected by a high-capacity optical network. The MCENs are considered as mini datacenters hosting both IT and Telecommunication (TLC) equipment, following the multi-access edge computing (MEC) model defined by ETSI, so that they can support the instantiation of VNFs. Following the MH guidelines, we assumed that these metro nodes are connected via bidirectional fiber links with transmission distances ranging from 5 km to 50 km. Each link contains two fibers with 20 wavelengths each at 100 Gbps/wavelength; in addition, each MCEN is also equipped with a set of 100 Gbit/s transponders. In terms of the nodes computational capacity required to host the VNFs related to each slice, we consider the server Intel® Xeon® Gold 6134 with 8 cores, processing capacity of 537.6 GFLOPS, maximum operating frequency of 3.7 GHz. In this work, we designed the 5G metro-core network as an undirected graph $G_S := (N_S, L_S)$, where N_S represents the set of all physical nodes (MCENs) and $L_S \subseteq N_S \times N_S$ represents the set of all physical links in the SN. A_N and A_L are the node attributes (CPU processing capability) and link attributes (bandwidth).

B. NETWORK SLICE REQUEST (NSR)

The NSR comprises a virtual topology composed of VNFs and virtual links with CPU and bandwidth requirements. In this work, we follow the separation between the 5G control plane and data plane VNFs [28], which includes the following VNFs: access/mobility management function (AMF), session management function (SMF), and user plane function (UPF).

We consider four types of NSRs based on 5G services: eMBB, mMTC, URLLC [21], and a generic slice that can be customized according to the user needs. Next, we present the details and requirements of the considered 5G NSs.

- eMBB: It requires high throughput (up to 20 Gbps) and computing resources, while the latency constraint is approximately 4 ms [25]. As a result, the deployment of eMBB requests aims to minimize the remaining resources of the physical nodes. Video streaming services and augmented reality applications belong to this class [23].
- mMTC: It includes a large number of connected online devices (1 million devices/km^2 [29]), such as IoT sensors. The deployment of mMTC requests aims to minimize bandwidth usage on physical links. These types of requests have plenty of connections; consequently, the requirement of computing resources is high and the demand for a low congestion rate.
- URLLC: this type or requests have strong latency constraints (e.g. 1 ms) and high availability requirements (e.g. 99.9%) [30]. The deployment of URLLC requests aim to minimize the delay. Autonomous driving services and eHealth applications are examples of this type of slice.

As for the SN, we denote each NSR as an undirected graph $G_V := (N_V, L_V)$, where N_V represents the set of all virtual nodes and $L_V \subseteq N_V \times N_V$ represents the set of all virtual links in the NSR. R_N and R_L are the node requests (CPU processing capability) and link requests (link bandwidth). Table 1 shows the mathematical notation of the SN and NSRs.

Notation	Description
N_S	Full set of physical nodes in SN
L_S	Full set of physical links in SN
A_N	Node attributes in SN
A_L	Link attributes in SN
N_V	Full set of virtual nodes in NSR
L_V	Full set of virtual links in NSR
R_N	Set of node requirements in NSR
R_L	Set of link requirements in NSR

TABLE 1: Mathematical notation for the SN and NSRs.

C. ADMISSION CONTROL MODULE

This section describes the proposed AC module based on the DRL. As mentioned in the Introduction, DRL is a subfield of machine learning that combines RL and deep learning. In particular, DRL agents can handle very large sets of input data and decide what actions to perform to optimize an objective. In this study, we implement the Advantage Actor-Critic algorithm (A2C) [15], which takes as input the current NSR and different information from the SN (i.e. state space), such as the number of metro-core nodes and links, the current CPU loads, bandwidth consumption, etc. Then, the agent learns to make decisions, such as accepting the NSR (i.e. action space) by optimizing the profit of the NSP. For this, we shaped a reward function that returns the goodness of action made by the agent. In the following subsections, we discuss the details of the A2C algorithm implementation.

1) A2C algorithm

Actor Critic algorithms have proven to be very efficient for problems with a large number of environmental states [31]. For instance, Md. Shirajum Munir et al. [32] designed a multi-agent A2C algorithm with the goal of providing an efficient energy scheduling scheme for a microgrid-powered MEC network. In particular, the objective is to reduce the gap between energy generation and demand estimation, where it can maximize the usage of renewable energy. Madyan Alsenwi et al. [33] proposed a novel approach combining optimization-theory based methods with the A2C algorithm to improve the performance of resource allocation of eMBB and URLLC traffic in wireless networks. These two research works show successful implementations of the A2C algorithm in different network scenarios. The objective of our work is to exploit the capability of the A2C algorithm to perform the AC task of NSs in the metro-core network scenario.

A2C is a specific type of actor-critic algorithm that belongs to the family of action-value functions. It combines two types of RL algorithms: policy-based and value-based.

Policy-based algorithms comprise agents that learn a policy, that is, a probability distribution of actions, by mapping the input state space to output actions. A policy can be represented as a rule used by the agent to select the correct action. It can be deterministic or stochastic, in which case it is usually denoted by π. Policy-based algorithms represent a policy as $\pi\left(a|s\right)$, where a is the action and s is the state.

On the other hand, value-based agents learn how to select actions based on the predicted value of the input state or action. Unlike policy-based algorithms, value-based agents aim to find a numerical representation of a state. In other words, the value is the expected reward E for state s under a policy π. The value function is denoted by $V^{\pi}(s)$ and it is represented as follows: $V^{\pi}(s) = E_{\pi}[r(t)\,|s]$.

As a result, action-value functions learn a value for the action instead of a state. The goal of the agent is to find an optimal policy $\pi^* : S \rightarrow A$ to maximize the expected reward. As such, we first define the value function $V^{\pi} : S \rightarrow \mathbb{R}$ which represents the expected value returned by following the policy π for each state $s \in S$. The value function V for policy π is defined in Eq. 1:

$$V^{\pi}(S) = E_{\pi}\left[r_t(s_t, a_t) + \gamma V^{\pi}(s_{t+1}) \mid s_0 = s\right] \quad (1)$$

Because the goal of the agent is to find the optimal policy π^*, an optimal action at each state can be expressed by the optimal value function: $V^*(s) = max_{a_t}\{E_{\pi}\left[r_t(s_t, a_t) + \gamma V^{\pi}(s_{t+1}) \mid s_0 = s\right]\}$. If we denote $Q(s,a) \triangleq r_t(s_t,a_t) + \gamma E_{\pi}\left[V^{\pi}(s_{t+1})\right]$ as the optimal Q-function for all state-action pairs, then the optimal value function can be written as follows: $V^*(s) = max_a\{Q^*(s,a)\}$. Therefore, the final goal is to find the optimal values of the Q-function, that is, $Q^*(s,a)$, for all state-action pairs, which can be done through iterative processes. In particular, the Q-function is updated according to the following rule:

$$\begin{aligned} Q_{t+1}(s,a) &= Q_t(s,a) \\ &\quad + \alpha_t\left[r_t(s,a) + \gamma max_{a'} Q_t(s,a') - Q_t(s,a)\right] \end{aligned} \quad (2)$$

In Eq. 2, the learning rate α_t defines the impact of new information on the existing Q-value. The algorithm then yields the optimal policy indicating an action to be taken at each state such that $Q^*(s,a)$ is maximized for all states in the state space, i.e., $\pi^*(s) = \arg\max_{\alpha} Q^*(s,a)$.

The traditional Q-learning is based on the concept that the agent knows the expected reward for each action at every step. However, it does not scale for problems with large states and action spaces. Therefore, DRL-based algorithms use deep learning to scale to decision-making problems that were previously intractable, that is, settings with high-dimensional state and action spaces. An actor-critic algorithm consists of two artificial neural networks: actor and critic. The actor network selects an action at each time step, and the critic network outputs the Q-value of a given input state. In other words, while the critic network learns which states are better or worse, the actor uses this information to explore good states and try to avoid bad states. The A2C algorithm is a specific version of the traditional actor-critic that exploits an advantage function that predicts the error of the agent [15]. The learning procedure of the $actor$ and $critic$ network is performed separately, and it uses gradient ascent to update both sets of weights in the corresponding networks. As time passes, the $actor$ is learning to produce better and better actions (it is starting to learn the policy), and the $critic$ is getting better and better at evaluating those actions.

In the following subsections, we provide a definition of the state space, action space, and reward function.

2) State space

A state is represented by the available resources in the substrate 5G metro-core network. To achieve efficient admission control, the agent needs sufficient information about the current state of the environment and the NSR characteristics. To achieve this, we define two feature vectors, $\varphi_S(G_S,t)$ and $\varphi_V(G_V,t)$, which are representative of the SN and NSR at a time t. The $\varphi_S(G_S,t)$ vector is composed by the following features:

- **Number of physical nodes (N')**: it is the number of physical nodes that have available CPU capacity at a time t. It is defined as follows: $N'(t) = \{\forall n \in N_S | A_N(n,t) > 0\}$
- **Number of physical edges (L')**: it is the number of physical links that have available bandwidth capacity at a time t. It is defined as follows: $L'(t) = \{\forall l \in L_S | A_L(l,t) > 0\}$
- **Average and standard deviation degree of physical nodes (ADp and SDp)**: the degree of a node is defined as the number of its neighboring edges. Average degree (ADp) is the average value of the degree of all nodes in the graph, i.e. $d(G,t) = \sum_i^{N'(t)} d(u_i,t)/N'(t)$ where

$d(u_i, t)$ denotes the degree of node u_i at a time t. While, the standard deviation degree (SDp) is the standard deviation of the degree of all nodes in the graph

- **Average and standard deviation of clustering coefficient (ACC and SCC)**: considering a node u, the clustering coefficient $cc_u(t)$ represents the likelihood that any neighbors of u are connected among each other at a time t. It is defined as: $cc_u(t) = \frac{2N_u(t)}{K_u(t)(K_u(t)-1)}$, where $N_u(t)$ is the number of links between the neighbors of u, while $K_u(t)$ is the degree of u. The average clustering coefficient (ACC) is the average of $cc_u(t)$ across all the nodes in the graph. The same applies for the standard deviation clustering coefficient (SCC)
- **Average and standard deviation of path length between physical nodes (APLp and SPLp)**: the average path length between physical nodes (APLp) is the average total path length at a time t between physical node u and every other physical node that is reachable from u. The same applies for the standard deviation of path length between physical nodes (SPLp)
- **Percentage of end points (PEP)**: it is defined as the percentage of nodes with degree equal to one
- **Free CPU (FCPU)**: it is the total available CPU capacity in the SN at a time t
- **Occupied CPU (OCPU)**: it is the total occupied CPU capacity in the SN at a time t
- **Free bandwidth (FB)**: it is the total available bandwidth capacity in the SN at a time t
- **Occupied bandwidth (OB)**: it is the total occupied bandwidth capacity in the SN at a time t
- **Total bandwidth (TB)**: it is the sum of the free and occupied bandwidth at a time t
- **Embedded nodes (EN)**: it is the number of embedded nodes at a time t
- **Embedded edges (EE)**: it is the number of embedded links at a time t

$\varphi_V(G_V, t)$ comprises the following features:

- **Number of virtual nodes (V')**: it is the number of virtual nodes of an NSR arrived at a time t
- **Number of virtual edges (T')**: it is the number of virtual links of an NSR arrived at a time t
- **Average degree of virtual nodes (ADv)**: the degree of a virtual node is defined as the number of its neighboring virtual edges. Average degree is the average value of the degree of all virtual nodes in the graph, i.e. $d(G, t) = \sum_i^{V'(t)} d(v_i, t)/V'(t)$ where $d(v_i, t)$ denotes the degree of virtual node v_i at a time t.
- **Average path length between virtual nodes (APLv)**: it is the averaged total path length at a time t between virtual node v and every other physical node that is reachable from v
- **CPU and bandwidth requests for NSRs (CPU_{NSR} and B_{NSR})**: the are defined as the CPU and bandwidth units requested by the specific type of slice
- NSR type: it defines the type of NSR, such as: generic, eMBB, URLLC and mMTC

As an example, Fig. 2 shows the embedding of a generic NSR made by 3 virtual nodes, with requirements on CPU and bandwidth, on the SN composed by 9 physical nodes; while Table 2 shows the values of the $\varphi_V(G_S, t)$ and $\varphi_V(G_V, t)$ state space vectors that describe the network at a time $t = 1$.

3) Action space

At each time step, the A2C-agent receives an NSR and decides whether to accept or deny the incoming slice. The set of actions is denoted by $a(t)$ and can assume a value equal to 0 (deny) or 1 (accept). The agent selects the action that maximizes the reward function.

4) Reward function

The reward function considers the resource efficiency of the SN in terms of revenue-to-cost-ratio (see section IV-A). We can improve the overall acceptance ratio if the agent predicts which NSR can be embedded in such a way that the resulting resource efficiency is maximum. If the agent rejects an NSR that cannot be embedded onto the SN, the VNE algorithm does not lose time trying to embed requests that are unfeasible. Therefore, the reward function depends on the success of the VNE algorithm. If the NSR can be embedded, the reward function is described by Eq. (3).

$$r_a = \begin{cases} \dfrac{revenue_{NSR_i}}{cost_{NSR_i}}, & \text{if } a = 1 \\ 0, & \text{if } a = 0 \end{cases} \tag{3}$$

On the contrary, if the VNE cannot embed the incoming NSR, the reward function is described in Eq. 4.

$$r_a = \begin{cases} \alpha, & \text{if } a = 1 \\ \beta, & \text{if } a = 0 \end{cases} \tag{4}$$

where α is a negative value between -1 and 0, and β is a positive value between 0 and 1. If α is closer to -1, the agent will be more cautious and will reject more incoming NSR; in contrast, if α is closer to 0 the agent will accept more requests. Algorithm 1 shows the complete AC procedure based on A2C.

D. VIRTUAL NETWORK EMBEDDING MODULE

The VNE module allocates the resources of each type of NSR to the SN. In this study, we extended two types of VNE algorithms introduced by the research work in [25].

The first is based on integer linear programming (ILP), a mathematical formulation model to solve optimization problems, such as the VNE. We start by defining different objective functions (one for each type of NSR) with the goal of minimizing or maximizing one (or more) specific metric, such as bandwidth and latency. Then, we define the constraints that consist of equations or inequalities that set upper and lower bounds on the variables of the model, for example, the CPU capability for each physical node. Compared to the work in [25], we modified the objective functions for eMBB

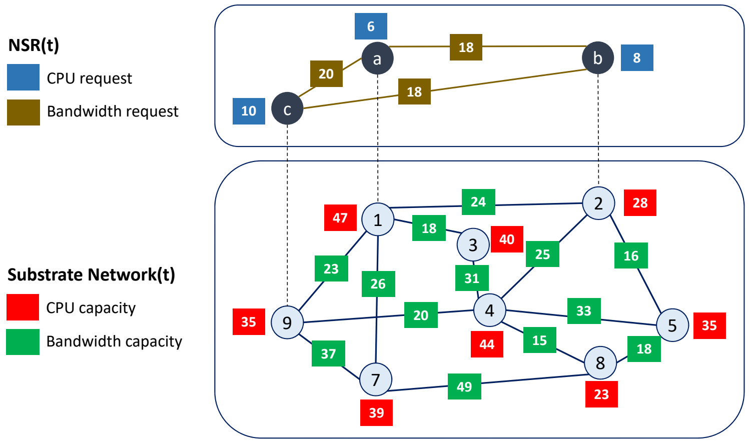

FIGURE 2: Example of a generic NSR embedding at a time $t = 1$

State space			
$\varphi_V(G_S,t)$	Value	$\varphi_V(G_V,t)$	Value
N'	9	V'	3
L'	12	T'	3
ADp	4.8	ADv	2
SDp	1.6	APLv	1
ACC	0.67	CPU_{NSR}	24
SCC	1.82	B_{NSR}	56
APLp	0.69	NSR type	Generic
SPLp	0.091		
PPE	0		
FCPU	291		
OCPU	0		
FB	335		
OB	0		
TB	335		
EN	0		
EE	0		

TABLE 2: State space at a time $t = 1$

and mMTC NSRs. Specifically, our ILP formulations tend to maximize the minimum physical resources left on the SN nodes and links. Consequently, a fair distribution of available resources is achieved throughout the SN to prevent some nodes, or links, from being used more than others.

However, the well-known VNE problem has been demonstrated to be NP-hard [34]; hence, exact solution methods suffer from scalability problems, unfeasible computational complexity, and running times. To address these issues, we developed a second set of algorithms based on heuristic methods. The goal is to obtain VNE solutions as close as possible to the optimal solutions in the shortest possible time. In particular, the run time of the VNE module is important to facilitate the rapid exploration/exploitation of the AC module and to speed up the embedding of the NSRs onto the SN. Compared to the work in [25], we developed the heuristic algorithms by taking into account specific requirements of the 5G NSRs. Therefore, we have considered the maximum CPU capacity of the SN nodes as the main feature when embedding the eMBB NSRs; then, we have used the minimum bandwidth available on the substrate nodes and the number of hops as the main features for the mMTC and URLLC NSRs respectively.

1) ILP-based algorithms

As stated above, we propose a mathematical formulation to accommodate different types of NSRs in the SN. It is based on a node-link formulation [22], which enables the synchronous embedding of virtual nodes and links by optimizing the allocation of physical network resources. Specifically, we define: 1) four objective functions (one for each NSR type); 2) a set of constraints shared by each type of NSR. Table 3 shows the variables and parameters used to develop the ILP.

a: Objective functions

The definition of the objective function is one of the major challenges in formulating an ILP. We define four different objective functions for the following NSR types:

- Generic NSR: This type of slice is used for those service requests that have no specific network constraints, as opposed to those that require specific network requirements such as eMBB, mMTC and URLLC. The idea is to allow slices of generic applications that do not need fixed allocations of resources in the SN. As such, the goal for a generic NSR is to take advantage of SN resources efficiently [25]. Therefore, the objective function minimizes the overall CPU and bandwidth. See Eq. (5)

$$\text{minimize} \sum_{v \in N_V} C^v \mu_p^v + \sum_{i,j \in L_V} B^{ij} u_{a,b}^{i,j} \tag{5}$$

- eMBB: As stated above, for this type of request, we have to minimize the CPU usage on physical nodes; hence, in this objective function, we maximize the minimum CPU capacity on physical nodes. See Eq. (6).

$$\text{maximize } \; T_{CPU} \tag{6}$$

$$\text{s.t } \; T_{CPU} \leq C_p(t) - \sum_{v \in N_V} C^v \cdot \mu_p^v, \quad \forall p \in N_S$$

  The constraint under Eq. (6) ensures that the CPU capacity assigned to the eMBB slice does not exceed the maximum capacity of the physical nodes.

- mMTC: For this type of request, we have to maximize the remaining resources of physical links; hence, in this objective function, we maximize the remaining bandwidth on physical links. See Eq. (7).

$$\text{maximize } \; T_{BW} \tag{7}$$

$$\text{s.t } \; T_{BW} \leq B_{ab}(t) - \sum_{i,j \in L_V} B^{i,j} \cdot u_p^v, \quad \forall a, b \in L_S$$

Algorithm 1: A2C-based AC algorithm

```
Data: Substrate network (SN), number of episodes,
      number of NSR per episode
Result: Updated AC Agent Networks
1  Initialize actor network parameters, $\theta_\pi$, with random parameters;
2  Initialize critic network parameters, $\theta_v$, with random parameters;
3  while $iteration \leq Num_{epochs}$ do
4      Generate $NSR_i$ according to an exponential distributed arrival rate $\lambda$, representing state $s_t$;
5      while $i \leq Num_{Requests}$ do
6          Get feature vectors $\varphi_S(G_S)$ and $\varphi_V(G_V)$;
7          Get the policy function $\pi_\theta(s_t, a_t)$ using actor network;
8          Sample actions from $\pi_\theta(s_t, a_t)$ to choose an action $a_t$;
9          if $a_t$ = 0 then
10             Reject $NSR_i$;
11             Verify rejection and calculate reward $r_t$;
12         else
13             Pre-accept $NSR_i$;
14             Calculate $r_t$ according to VNE output;
15             if VNE output = True then
16                 Start life-cycle of the $NSR_i$;
17             else
18                 Reject $NSR_i$;
19             end
20         end
21         Get feature vectors $\varphi_S(G_S)$ and $\varphi_V(G_V)$, representing state $s_{t+1}$;
22         if $i = Num_{Requests}$ then
23             R = 0;
24         else
25             $R \leftarrow r_t + \gamma R$;
26         end
27         Calculate value $V(s_t)$ using the critic network;
28         Calculate value $V(s_{t+1})$ using the critic network;
29         Calculate advantage value $A(s_t, a_t)$;
30         Calculate policy gradient $\delta\theta_\phi$
31         Calculate value gradient $\delta\theta_v$
32         Update actor network parameters using policy gradient $\delta\theta_\phi$ ;
33         Update critic network parameters using value gradient $\delta\theta_v$;
34     end
35 end
```

The constraint under Eq. (7) ensures that the bandwidth capacity assigned to the mMTC slice does not exceed the maximum bandwidth of the physical links.

- URLLC: The objective function for this type of requests is the minimization of the number of physical links used to embed virtual links of the NSR. See Eq. (8).

$$\text{minimize} \sum_{i,j \in L_V} u^{i,j}_{a,b} \tag{8}$$

b: Constraints

To ensure a feasible embedding between the virtual nodes/links of the NSR and the physical SN resources, we define a set of constraints shared by each NSR type.

1) Assignment of virtual nodes to physical nodes: Eq. (9) ensures that each virtual node is assigned to one physical node.

$$\sum_{p \in N_S} \mu^v_p = 1, \quad \forall v \in N_V \tag{9}$$

2) Assignment of physical nodes to virtual nodes: Eq. (10) ensures that each physical node can embed at a maximum of one virtual node per NSR.

$$\sum_{v \in N_V} \mu^v_p \leq 1, \quad \forall p \in N_S \tag{10}$$

3) CPU capability conservation: Eq. (11) ensures that the available CPU capacity of each physical node at time step t is not exceeded.

$$\sum_{v \in N_V} C^v \cdot \mu^v_p \leq C_p(t), \quad \forall p \in N_S \tag{11}$$

4) Bandwidth capability conservation: Eq. (12) ensures that the available bandwidth capacity of each physical link at time step t is not exceeded.

$$\sum_{i,j \in L_V} B^{i,j} \cdot u^{i,j}_{a,b} \leq B_{ab}(t), \quad \forall a,b \in L_S \tag{12}$$

5) Multi-commodity flow conservation with node-link formulation: Eq. (13) optimizes the mapping of virtual links and virtual nodes. The multi-commodity flow constraint is applied within a node-link formulation [35].

$$\sum_{a,b \in L_S} (u^{i,j}_{a,b} - u^{i,j}_{b,a}) = \mu^i_a - \mu^j_a, \quad \forall p \in N_S, \forall i,j \in L_V \tag{13}$$

6) Integrity constraints: Eq. (14) defines the binary variables.

$$\begin{aligned} \mu^v_p &\in \{0,1\}, \forall p \in N_S, \forall v \in N_V \\ u^{ij}_{ab} &\in \{0,1\}, \forall a,b \in L_S, \forall i,j \in L_V \end{aligned} \tag{14}$$

2) Heuristic-based algorithms

The VNE problem can be divided into two sub-problems: virtual node mapping (VNoM) and virtual link mapping (VLiM). Both sub-problems can be solved in a coordinated or uncoordinated fashion [35]. The coordination of the VNE can be achieved in one stage by simultaneously solving the VNoM and VLiM at the same time. The uncoordinated VNE is performed by solving each subproblem separately. In this work, we implement the uncoordinated method because we have different types of requests with different embedding

	Notation	Description
Variables	μ_p^v	Binary variable, 1 if virtual node v in NSR, is mapped to physical node p in SN
	u_{ab}^{ij}	Binary variable, 1 if virtual link (i,j) in NSR, is mapped to physical link (a,b) in SN
Parameters	$C_p(t)$	Free CPU resources of physical node p at step time t
	$B_{ab}(t)$	Bandwidth capability resources of physical link between node a and node b at time t
	C^v	CPU resources requirement of virtual node v
	B^{ij}	Bandwidth resources requirement of virtual link (i,j)
Sets	N_S	Set of all physical nodes present in SN
	N_V	Set of all virtual nodes present in NSR
	L_S	Set of all physical links present in SN
	L_V	Set of all virtual links present in NSR

TABLE 3: ILP mathematical notations

requirements; some of them focus on node resources, while others focus on link resources. Hence, we solve the VNE problem using different algorithms for node mapping and link mapping. The order in which these algorithms are applied to the embedding of the NSR onto the SN depends on the type of NSR.

a: Heuristic algorithm for General NSRs

Algorithm 3 presents the VNE procedure for a generic NSR. It starts by solving VNoM as follows:

1) First, it computes the node resource (NR) and the node importance (NI) metrics for both virtual and physical nodes. NR represents the available resources of each node, and is defined by Eq. 15. NI aims to score the nodes based on their resources and centrality in the network. It is defined by Eq. 16.

$$NR(i) = CPU(i) \cdot \sum_{l \in s(i)} BW(l) \tag{15}$$

$CPU(i)$ is the CPU resource capacity of node i; $s(i)$ represents the set of links that are directly connected to node i; $BW(l)$ represents the bandwidth capacity of link l.

$$NI(i) = NR(i) \cdot \left(\frac{d_i^{'} \cdot b_i^{'}}{2} \right) \tag{16}$$

$NR(i)$ is the node resource metric defined by Eq. 15; $d_i^{'}$ is the normalized degree of node i, and $b_i^{'}$ is the normalized betweenness centrality of node i. The latter is defined as the number of shortest paths between two nodes that pass through node i.

2) Once the algorithm computes the NR and NI metrics, it sorts the virtual nodes using Algorithm 2. Each node $v \in NSR_i$ is sorted in a non-increasing order using the breadth-first-search (BFS) algorithm [36] by exploiting the NI values of each node. The NI metric allows the nodes to by classified according to the number of shortest paths that can pass-through each node. BFS algorithm gives priority to nodes with higher NI; as such, virtual nodes $v \in NSR_i$ are mapped onto the most important physical nodes.

3) If the VNoM procedure is successful, the algorithm proceeds with the VLiM procedure. The link mapping procedure is based on the shortest path between embedded physical nodes, and in particular, we use the Floyd-Warshal algorithm [37]. For each virtual link $\in NSR_i$, it removes the physical links that do not meet the bandwidth requirements. Then, it determines the physical nodes where the source and target virtual nodes of the virtual link are mapped. Finally, it maps the virtual link onto the physical links in the physical shortest path calculated using the Floyd-Warshal algorithm.

```
Algorithm 2: Virtual nodes sorting
   Data: $N_v$: Set of virtual nodes in NSR.
   Result: $N_v^i$: sequence of sorted virtual nodes.
1  Calculate the NI of each virtual node;
2  Sort virtual nodes according to their NI value in a
     non-increasing order;
3  Select the virtual node with highest NI value as the
     root node;
4  Traverse the NSR graph using BFS algorithm, and get
     the BFS tree T;
5  Sort virtual nodes in each layer of T according to the
     NI value in a non-increasing order.
6  Return $N_v^i$
```

b: Heuristic algorithm for eMBB NSRs

This heuristic aims to minimize the CPU usage on physical nodes. Hence, we first run the VNoM procedure and then the VLiM procedure. The virtual nodes were sorted using Algorithm 2. The physical node in the SN with the highest CPU capacity was selected to map each virtual node of the incoming NSR. If the VNoM succeeds, we run the Floyd-Warshal algorithm for the VLiM procedure with the goal of embedding the virtual link onto the physical links in the SN shortest path. See Algorithm 4.

Algorithm 3: VNE heuristic algorithm for Generic NSRs

```
Data: Incoming $NSR_i$ and $SN$
Result: Success or failure of the VNE procedure
1  Sort physical nodes $p \in N_S$ of $SN$ in a non-increasing
     order according to its NI;
2  Sort virtual nodes $v \in N_{V_i}$ of $NSR_i$ according to
     Algorithm 2;
3  for $v \in N_{V_i}$ do
4      if $v$ is the root node then
5          Map $v$ onto the physical node with highest NI;
6      else
7          Find parent virtual node P of virtual node $v$ in
             T;
8          Find physical node PP in which P is mapped;
9          Set C the adjacent physical nodes of PP;
10         Map $v$ onto the physical node with highest NI
             in C;
11     end
12 end
13 Sort virtual links $(i, j) \in L_{V_i}$ of $NSR_i$ in a
     non-increasing order according to their bandwidth
     requirement;
14 for $(i, j) \in L_{V_i}$ do
15     Remove the physical links that can not meet the
         bandwidth requirement;
16     Find physical node a where virtual node i is
         mapped;
17     Find physical node b where virtual node j is
         mapped;
18     Find the physical shortest path between a and b by
         using Floyd Warshall algorithm;
19     Map vitual link $(i, j)$ onto the physical shortest
         path.
20 end
```

c: Heuristic algorithm for mMTC NSRs

This heuristic maximizes the remaining bandwidth on physical links; it first runs the VLiM and then the VNoM procedure. The algorithm sorts the virtual links of the NSR in non-increasing order according to their bandwidth requirements. The goal is to first embed virtual links with higher requirements. Then, it calculates all possible paths that can allocate each virtual link and selects the best fitting path in the SN. The best-fitting path is selected according to the following procedure: 1) it selects the physical link with the minimum BW resources for each possible path, and 2) it selects the path where the minimum link BW is maximum among all possible paths. Then, the physical links on the best-fitting path are selected to map the virtual link. If the VLiM is successful, we select the source and target physical nodes in the best-fitting path of each virtual link to map the virtual nodes of the incoming NSR. See Algorithm 5.

Algorithm 4: VNE heuristic algorithm for eMBB NSRs

```
Data: Incoming $NSR_i$ and $SN$
Result: Success or failure of the node mapping
          procedure
1  Sort virtual nodes $v \in N_{V_i}$ of $NSR_i$ according to
     Algorithm 2;
2  for $v \in N_{V_i}$ do
3      if $v$ is the root node then
4          map$v$ onto the physical node with highest
             CPU capacity resources;
5      else
6          Find parent virtual node P of virtual node $v$ in
             T;
7          Find physical node PP in which P is mapped;
8          Set C the adjacent physical nodes of PP;
9          Map$v$ onto the physical node in C with highest
             CPU capacity resources;
10     end
11 end
12 Sort virtual links $(i, j) \in L_{V_i}$ of $NSR_i$ in a
     non-increasing order according to their bandwidth
     requirement;
13 for $(i, j) \in L_{V_i}$ do
14     Remove the physical links that can not meet the
         bandwidth requirement;
15     Find physical node a where virtual node i is
         mapped;
16     Find physical node b where virtual node j is
         mapped;
17     Map vitual link $(i, j)$ onto the physical shortest
         path between a and b;
18 end
```

d: Heuristic algorithm for URLLC NSRs

Finally, the URLLC request must be deployed with a specific delay requirement. To do so, the heuristic algorithm must minimize the end-to-end delay between sources and destinations; in other words, it will use fewer physical links. As such, it first runs the VLiM and then the VNoM procedure. Similar to the previous algorithm, we first sort the virtual links of the incoming NSRs in a non-increasing order according to their bandwidth requirements. Then, it calculates all possible paths that can allocate each virtual link and selects the best-fitting path in the SN. The best-fitting path is the one with fewer hops in the SN. If VLiM is successful, we select the source and target physical nodes in the best fitting path of each virtual link to map the virtual nodes of the incoming NSR. See Algorithm 6.

IV. PERFORMANCE EVALUATION

This section introduces the performance evaluation of the proposed ES framework. First, we present the metrics used for assessing the performance. Afterwards, we show the

Algorithm 5: VNE heuristic algorithm for mMTC NSRs

```
Data: Incoming $NSR_i$ and $SN$
Result: Success or failure of the link mapping procedure
1  Sort virtual links $(i, j)$ ∈$L_{V_i}$ of $NSR_i$ in a
     non-increasing order according to their bandwidth
     requirement;
2  for $(i, j)$ ∈$L_{V_i}$ do
3      Find all physical candidate paths $CP$ where the
         virtual link $(i, j)$ can be mapped;
4      for $p \in CP$ do
5          Find the physical link $l \in p$ with the minimum
             bandwidth resources.
6      end
7      Select the path $bp \in CP$ where $l$ is maximum;
8      Map the the virtual link $(i, j)$ onto path $bp$;
9      Map virtual node $i$ onto the the physical source
         node of the $bp$ path;
10     Map virtual node $j$ onto the the physical target
         node of the $bp$ path;
11 end
```

Algorithm 6: VNE heuristic algorithm for URLLC NSRs

```
Data: Incoming $NSR_i$ and $SN$
Result: Success or failure of the link mapping procedure
1  Sort virtual links $(i, j)$ ∈$L_{V_i}$ of $NSR_i$ in a
     non-increasing order according to their bandwidth
     requirement;
2  for $(i, j)$ ∈$L_{V_i}$ do
3      Find all physical candidate paths $CP$ where the
         virtual link $(i, j)$ can be mapped;
4      Select the path $bp \in CP$ whit the minimum
         number of hops;
5      Map the the virtual link $(i, j)$ onto path $bp$;
6      Map virtual node $i$ onto the the physical source
         node of the $bp$ path;
7      Map virtual node $j$ onto the the physical target
         node of the $bp$ path;
8  end
```

performance of the VNE module by comparing the ILP-based and heuristic-based algorithms separately; and then we present the evaluation of the entire ES framework with the AC module based on DRL. Finally, we discuss the complexities of the algorithms and the results of this study.

A. PERFORMANCE METRICS

The VNE problem is defined as a mapping problem in which VNE algorithms map an NSR (G_V) to an SN (G_S). The mapping procedure of an NSR onto the SN is only valid if the full set of virtual nodes N_V is mapped onto a subset of substrate nodes $N_S^{'}$ ∈N_S, and all virtual links L_V connecting virtual nodes are mapped onto a subset of substrate nodes $L_S^{'}$ ∈L_S. All virtual node and link requests in the G_V must be satisfied. The following sections introduce: acceptance ratio, resource efficiency and run time metrics.

1) Acceptance Ratio (AR)

The acceptance ratio is the ratio between the number of requests that have been successfully mapped and the total number of requests arrived to the AC module [25]. It is defined in Eq. (17):

$$AR = \frac{\sum_{t=0}^{T} NSR_{accepted}(t)}{\sum_{t=0}^{T} NSR(t)} \tag{17}$$

2) Resource Efficiency (RE)

Resource efficiency is defined as the revenue-to-cost-ratio [25]. When an NSR is accepted, the mapping revenue can be defined as the sum of its nodes capacity and link bandwidth requirements. The cost can be defined as the sum of the node and link bandwidth resources of the SN. It is defined in Eq. (18):

$$RE = \frac{\sum_{n \in N_V} R_N(n) + \sum_{l \in L_V} R_L(l)}{\sum_{n \in N_V} R_N(n) + \sum_{l \in L_V} R_L(l) * hop(l)} \tag{18}$$

$R_N(n)$ represents the CPU capacity requirement of virtual node n; $R_L(l)$ represents the bandwidth requirement of virtual link l; and $hop(l)$ represents the path length of link l in which the virtual link l is mapped onto the SN.

3) Run Time (RT)

VNE algorithms experience a trade-off between performance in terms of resource efficiency and run time. ILP-based algorithms can guarantee optimal VNE solutions but suffer from scalability issues and long run times. In contrast, heuristic-based algorithms provide suboptimal but much faster VNE solutions. This trade-off must be considered in the case of real-time network slice deployment. In this work, we evaluate the run time of the proposed ES framework as a metric for real-time network services.

B. EXPERIMENTS SETUP

The proposed ES framework was developed using Python version 3.6.7. We used Ubuntu 18.04 LTS with an Intel Core i7 CPU and 8 GB RAM. We used the Barabasi-Albert algorithm [38] to generate topologies for the SN and NSRs. In this paper, evaluations are given for 50-MCENs SN topology and 3-virtual node topologies for generic, eMBB, URLLC and mMTC [28] [39]. Both physical processing units capacities and bandwidth unit are uniformly distributed between 20 and 50, $U(20, 50)$. NSR processing units and bandwidth units are distributed according to the NSR type as follows:

- The CPU requirement is uniformly distributed between 5 and 15 units for each NSR types

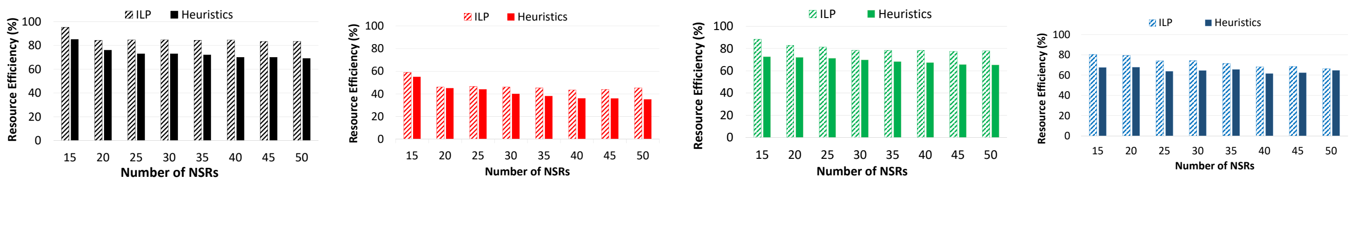
(a) Generic NSR (b) eMBB NSR (c) mMTC NSR (d) URLLC NSR

FIGURE 3: VNE module: Resource efficiency for each type of NSR.

- The Bandwidth is distributed as follows: 5 units for Generic NSR, 4 units for eMBB NSR, 3 units for URLLC NSR and 2 units for mMTC NSR

The NSRs of each type arrive with an exponentially distributed arrival rate λ = 1/10, 3/10, 5/10 and stay for an exponentially distributed lifetime with a mean value of 100 s.

The A2C algorithm comprises an actor network with an input layer, a recurrent neural network (RNN) layer, two dense layers, and finally, a softmax layer. Moreover, the critic network comprises an input layer, a recurrent neural network (RNN) layer, two dense layers, and a single unit layer. The actor network returns the policy $\pi_\theta(s_t, a_t)$, a probability distribution of two possible actions (accept or reject) for the incoming NSR. The critic network returns the value $V_\theta(s_t, \theta_v)$. Both actor and critic networks have the same input and hidden layers that are described as follows:

1) The input layer takes the concatenation of the feature vectors $\varphi_S(G_S, t)$ and $\varphi_V(G_V, t)$.
2) The output of the input layer is forwarded to the RNN layer. It is implemented using a long short term memory (LSTM) [40] with 200 nodes and an hyperbolic tangent activation function.
3) Two dense layers with 128 and 64 nodes, respectively, are implemented. Both dense layers implement a rectified linear unit (ReLU) activation function.

Then, we set the A2C algorithm parameters as follows: (learning rate) $\alpha = 0.0003$, (discount rate) $\gamma = 0.99$, (maximum exploration factor) $\epsilon_{max} = 1$. Training starts at 400 steps, mini-batch size to 20 samples and actor-critic updates every 150 steps. The hyper-parameters of the A2C algorithm were chosen by adopting greedy layer-wise training [41]. We obtained 95% confidence level of 50 repetitions of the experiments.

C. RESULTS

This section presents the results of the proposed ES framework. We first show the performance of the VNE module in stand-alone mode, by evaluating the RE and RT for the ILP-based and heuristic algorithms introduced in sections III-D1 and III-D2.

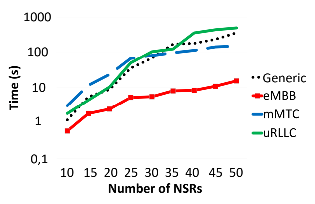
FIGURE 4: ILP-based algorithm run time

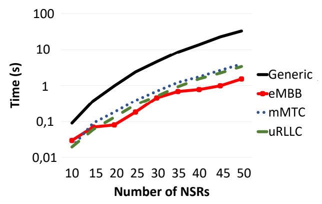
FIGURE 5: Heuristic-based algorithms run time

Subsequently, we evaluate the performance of the entire ES made by the AC and VNE modules, the last equipped with the heuristic-based algorithms. In this evaluation, we compared two AC algorithms: the proposed A2C implementation and a state of the art DRL algorithm called Deep Q-learning (DQN) [42]. We set the DQN algorithm as follows: 1) policy network is an Artificial Neural Network (ANN) with 2 hidden layers made by 128 artificial neurons with a softmax activation function at the output layer; 2) discount factor equal to 0.99; 3) learning rate equal to 0.0005; 4) exploration fraction equal to 0.1.

1) VNE module: ILP-based and heuristic-based algorithms comparison

Figure 3 compares the resource efficiency results of the generic, eMBB, mMTC, and URLLC NSR types, respectively. We start by generating 15 NSRs until 50 NSRs of each type. As shown in the figure, the RE shows a small difference between the heuristic and the ILP-based algorithms. Specifically, it is -12% for generic, -5.8% for eMBB, -11.3% for mMTC and -8% for URLLC. These results confirm that there is minimal difference between the solutions derived from ILP-based algorithms and those based on heuristics. Moreover, we can assess that the algorithms that perform better are those in which the objective function works more in saving link resources (generic, URLLC and mMTC) than node resources (eMBB).

Figure 4 and 5 compares the run time (in *Log* scale) of the different types of requests for both the ILP and heuristic algorithms. Using the ILP method, the eMBB request is embedded significantly faster than the other request types, with a mean RT of 15 seconds compared to the other NSR

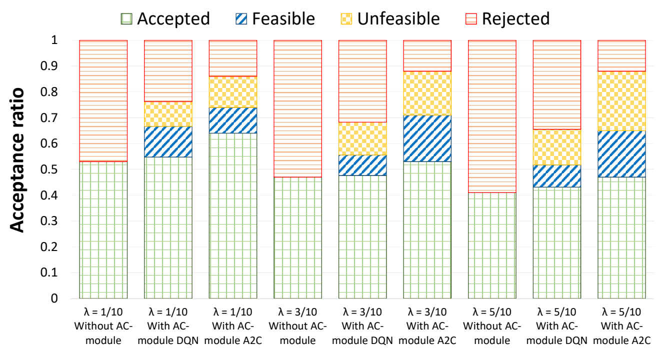

FIGURE 6: Acceptance ratio of incoming NSRs with and without AC module.

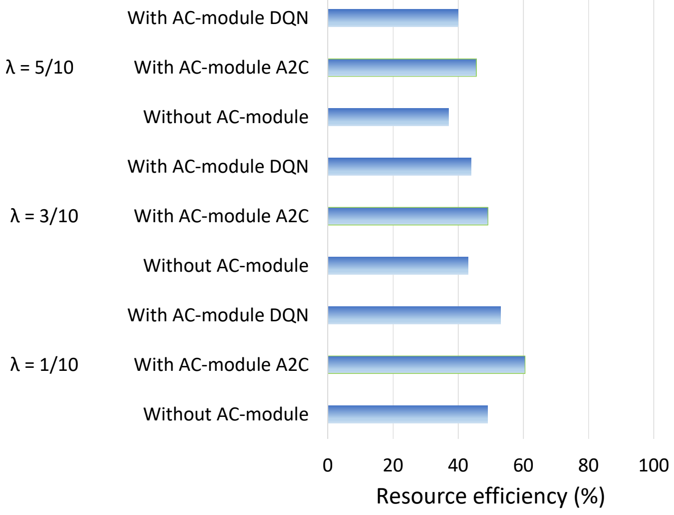

FIGURE 7: Resource efficiency with and without AC module.

types that require an average RT of $493, 33$ seconds. That is because it searches only for a set of candidate nodes ($|N_S| = 50$) for each virtual node. In contrast, the other types of requests take into account also the set of candidate paths. This leads to less time consumption when embedding an eMBB request type.

On the other hand, using the heuristic algorithms, the embedding procedure of a generic NSR type takes more time with respect to the others (approximately 20 s more) because the node mapping and link mapping procedures of Algorithm 3 are the longest in terms of time complexity. Furthermore, heuristic algorithms can embed all NSR types with a mean time of 7 s, while the ILP-based algorithm takes almost $372, 48$ s. Thus, given the high performance of the heuristic algorithms in terms of both RE and RT, we can use them as VNE algorithms within the VNE module of the proposed ES framework.

2) Proposed ES framework: AC module + VNE module

After assessing the performance of the ILP and heuristic-based algorithms, this section shows the performance of the entire ES framework, as shown in Fig. 1, made by the AC module, equipped with two DRL algorithms (A2C and DQN), and the VNE module, equipped with the heuristic algorithms. Figure 6 shows the acceptance ratio of incoming NSRs with and without the proposed AC module according to the selected DRL algorithm. Without admission control means that every time a NSR arrives, it is embedded by the VNE module. Considering Fig. 6, the four labels represent:

- Accepted: overall accepted NSRs
- Feasible: NSRs rejected by the AC module that can be embedded when they arrive at the AC module. This amount of NSR was previously refused in order to optimize SN's resources
- Unfeasible: NSRs rejected by the AC module that cannot be embedded. This means that the AC module learned to successfully pre-reject the NSRs that cannot be embedded onto the SN
- Rejected: overall rejected NSRs by the VNE module

The results show that for increasing values of λ, the number of accepted NSRs decreases. When NSRs are generated with high arrival rates, the physical resources are occupied much faster than when they are deployed with shorter arrival rates, leading to a higher number of rejected NSRs.

The number of rejected requests indicates the number of requests that the VNE module tried to implement in the SN but failed due to scarce resources left. We can see how this amount is greatly reduced when we enable the AC module based on A2C (37%). We can notice that the DRL-based algorithm has learned to reject in advance the NSRs that cannot be implemented in the SN. The latter is indicated by unfeasible and feasible labels. The former identifies NSRs that cannot be implemented when they arrive. In fact, the AC module (A2C) has learned to reject them. In contrast, NSRs in blue could have been implemented on the SN because they are eligible when they arrive. However, the AC module (A2C) "preferred" to reject them to optimize the total RE of the SN. Thanks to the AC module (A2C), our ES framework accepts 15% more NSRs than the case in which the AC module (A2C) is disabled. On the other side, the AC module based on DQN did not show good results. In particular, the agent's policy network did not converge by showing almost the same results as the case in which the AC module is not enabled.

Table 4 shows a detailed view on the acceptance ratio for each type of NSR with an AC module based on A2C. In this case, the amount of requests that are unfeasible and feasible from Figure 6 are considered rejected in Table 4. We can see that embedding eMBB slices is difficult in a dynamic environment where the types of incoming NSRs are unknown. This is due to the fact that eMBB slices require high computational resources, in terms of CPU units, compared to other types of slices.

Figure 7 presents the overall resource efficiency once all

NSR type $\lambda = 1/10$	Without AC module		With AC module (A2C)	
	Accepted	Rejected	Accepted	Rejected
Generic	0.15	0.1	0.17	0.08
eMBB	0.08	0.17	0.13	0.12
URLLC	0.14	0.11	0.18	0.07
mMTC	0.16	0.09	0.17	0.08
	Total = 0.53	Total = 0.47	Total = 0.65	Total = 0.35

NSR type $\lambda = 3/10$	Without AC module		With AC module (A2C)	
	Accepted	Rejected	Accepted	Rejected
Generic	0.14	0.11	0.12	0.13
eMBB	0.07	0.18	0.09	0.16
URLLC	0.16	0.09	0.17	0.08
mMTC	0.1	0.15	0.15	0.1
	Total = 0.47	Total = 0.53	Total = 0.53	Total = 0.47

NSR type $\lambda = 5/10$	Without AC module		With AC module (A2C)	
	Accepted	Rejected	Accepted	Rejected
Generic	0.11	0.14	0.12	0.13
eMBB	0.07	0.18	0.07	0.18
URLLC	0.13	0.12	0.15	0.1
mMTC	0.1	0.15	0.13	0.12
	Total = 0.41	Total = 0.59	Total = 0.47	Total = 0.53

TABLE 4: Acceptance ratio with and without AC module (A2C) for each type of NSR.

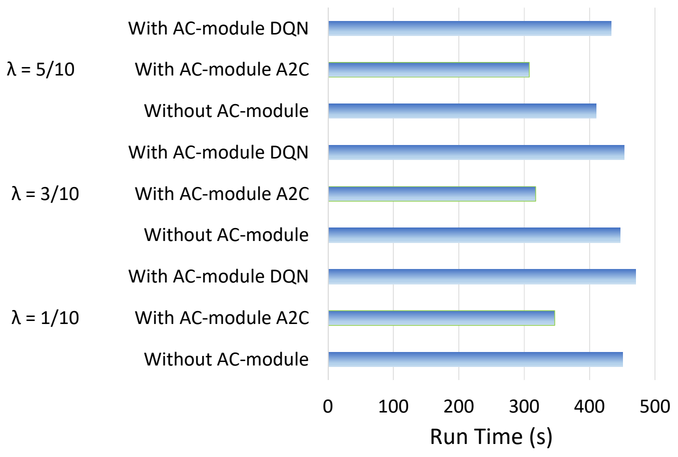

FIGURE 8: Overall Run-time with and without AC module.

NSRs are embedded. It can be observed that the resource efficiency decreases while λ increases. This is because with increasing λ, node resources are generally scarce, leading to embedding virtual links onto paths with a higher number of physical links as non-adjacent physical nodes are selected, resulting in higher costs and lower resource efficiency. Moreover, for all λ values, it can be seen that the overall resource efficiency increases when the AC module is implemented, both in the A2C case and in the DQN case. Obviously, given the poor performance of the DQN algorithm in accepting slices, the efficiency is lower than in the case of the A2C.

Finally, Figure 8 shows the run time with and without AC module. We start measuring the time since the arrival of each NSR until the final embedding by the VNE module. We can observe that the overall run time decreases when the AC module based on A2C is implemented. This result shows the performance of the AC module (A2C) in terms of the filtering capabilities. In other words, it filters NSRs that are either unfeasible or lead to lower resource efficiency. Consequently, the VNE module does not waste time in processing NSRs that cannot be embedded. On the other hand, the DQN based AC module takes much longer than the other cases since it does not converge towards a stable solution, i.e. it is unable to maximize the overall return.

Algorithm	**Complexity ($\mathcal{O}$)**
1	$\mathcal{O}(Num_{epochs} \times Num_{NSRs})$
3	$\mathcal{O}(N_v + L_v)$
4	$\mathcal{O}(N_v + L_v)$
5	$\mathcal{O}(L_v \times L_s \times N_s)$
6	$\mathcal{O}(L_v)$

TABLE 5: Complexity analysis

D. COMPLEXITY ANALYSIS

Table 5 summarizes the complexity of each algorithm proposed by this work. Algorithm 1 aims at training a DRL-based agent to maximize the number of admitted slices into the SN. RL is generally known to be highly data intensive, and the amount of state-action space determines the computational complexity and processing time. For instance, an environment involving $N \times M$ state matrix, the computational complexity may be $\mathcal{O}(NM)$. Indeed, if the state matrix becomes too large the complexity may grow exponentially [43] and the computational cost becomes too high. Besides, DRL-based agents perform action-value approximations through deep learning techniques, such as artificial neural networks. Indeed, the computational complexity can be expressed as the multiplication between the number of epochs to train the agent and the number of NSRs to process: $\mathcal{O}(Num_{epochs} \times Num_{NSRs})$.

Algorithm 3 and 4 have the same computational complexity since both perform first a virtual node mapping and then a virtual link mapping, hence the complexity is $\mathcal{O}(N_v + L_v)$.

Algorithm 5 is more computationally intensive since it calculates all possible paths that can allocate each virtual link and selects the best fitting path in the SN. The complexity is equal to $\mathcal{O}(L_v \times L_s \times N_s)$.

Algorithm 6 minimizes the end-to-end delay between sources and destinations by reducing the number of used physical links. As such, the complexity is related to the virtual link mapping: $\mathcal{O}(L_v)$.

V. CONCLUSION

This work introduced a novel framework to jointly perform AC and VNE of 5G NSs. The AC module is based on the DRL A2C algorithm, which is able to select the most profitable NSRs to accommodate onto the SN. The VNE module is based on a set of heuristic algorithms (validated by

ILP mathematical formulations) with the aim of embedding each type of 5G NSRs according to their requirements. Our proposed ES framework achieved up to 15% more accepted NSRs with respect to the case in which the AC module is disabled. Moreover, while accepting more requests, our ES framework can increase the resource efficiency and decrease the run time of the admission control and embedding procedure. In future work, we will enrich the VNE module with algorithms based on DRL. With this aim, we plan to build a comprehensive ES made by DRL-based agents, both for AC and VNE, to further improve the acceptance ratio and resource efficiency in the 5G metro-core network.

REFERENCES

[1] "View on 5g architecture," 5GPPP Architecture Working Group, Tech. Rep., 2021.

[2] M. Series, "Imt vision–framework and overall objectives of the future development of imt for 2020 and beyond," Recommendation ITU, vol. 2083, p. 0, 2015.

[3] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal, "Nfv: state of the art, challenges, and implementation in next generation mobile networks (vepc)," IEEE Network, vol. 28, no. 6, pp. 18–26, 2014.

[4] M. Chiosi, D. Clarke, P. Willis, A. Reid, J. Feger, M. Bugenhagen, W. Khan, M. Fargano, C. Cui, H. Deng et al., "Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action," in SDN and OpenFlow World Congress, vol. 48. sn, 2012.

[5] F. Z. Yousaf, M. Bredel, S. Schaller, and F. Schneider, "Nfv and sdn—key technology enablers for 5g networks," IEEE Journal on Selected Areas in Communications, vol. 35, no. 11, pp. 2468–2478, 2017.

[6] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," IEEE Communications Surveys Tutorials, vol. 15, no. 4, pp. 1888–1906, 2013.

[7] 3GPP, "document r1-1612306," 3GPP TSG RAN WG1 Meeting 87, Tech. Rep., 2016.

[8] A. K. Bairagi, M. S. Munir, M. Alsenwi, N. H. Tran, S. S. Alshamrani, M. Masud, Z. Han, and C. S. Hong, "Coexistence mechanism between embb and urllc in 5g wireless networks," IEEE Transactions on Communications, vol. 69, no. 3, pp. 1736–1749, 2020.

[9] M. Alsenwi, N. H. Tran, M. Bennis, A. K. Bairagi, and C. S. Hong, "embb-urllc resource slicing: A risk-sensitive approach," IEEE Communications Letters, vol. 23, no. 4, pp. 740–743, 2019.

[10] D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, and X. Costa-Pérez, "A machine learning approach to 5g infrastructure market optimization," IEEE Transactions on Mobile Computing, vol. 19, no. 3, pp. 498–512, 2019.

[11] M. R. Raza, A. Rostami, L. Wosinska, and P. Monti, "A slice admission policy based on big data analytics for multi-tenant 5g networks," Journal of Lightwave Technology, vol. 37, no. 7, pp. 1690–1697, 2019.

[12] B. Han, V. Sciancalepore, X. Costa-Perez, D. Feng, and H. D. Schotten, "Multiservice-based network slicing orchestration with impatient tenants," IEEE Transactions on Wireless Communications, vol. 19, no. 7, pp. 5010–5024, 2020.

[13] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[14] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," 2017.

[15] 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. PMLR, 2016, pp. 1928–1937.

[16] (2019) Metro-haul project. [Online]. Available: https://metro-haul.eu/

[17] R. Challa, V. V. Zalyubovskiy, S. M. Raza, H. Choo, and A. De, "Network slice admission model: Tradeoff between monetization and rejections," IEEE Systems Journal, vol. 14, no. 1, pp. 657–660, 2019.

[18] M. R. Raza, C. Natalino, P. Öhlen, L. Wosinska, and P. Monti, "Reinforcement learning for slicing in a 5g flexible ran," Journal of Lightwave Technology, vol. 37, no. 20, pp. 5161–5169, 2019.

[19] V. Sciancalepore, X. Costa-Perez, and A. Banchs, "Rl-nsb: Reinforcement learning-based 5g network slice broker," IEEE/ACM Transactions on Networking, vol. 27, no. 4, pp. 1543–1557, 2019.

[20] A. Blenk, P. Kalmbach, P. van der Smagt, and W. Kellerer, "Boost online virtual network embedding: Using neural networks for admission control," in 2016 12th International Conference on Network and Service Management (CNSM), 2016, pp. 10–18.

[21] S. M, "Imt vision–framework and overall objectives of the future development of imt for 2020 and beyond," 2015.

[22] M. Melo, S. Sargento, U. Killat, A. Timm-Giel, and J. Carapinha, "Optimal virtual network embedding: Node-link formulation," IEEE Transactions on Network and Service Management, vol. 10, no. 4, pp. 356–368, 2013.

[23] K. Ludwig, A. Fendt, and B. Bauer, "An efficient online heuristic for mobile network slice embedding," in 2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), 2020, pp. 139–143.

[24] A. Fendt, C. Mannweiler, K. Ludwig, L. C. Schmelz, and B. Bauer, "End-to-end mobile network slice embedding leveraging edge computing," in NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium, 2020, pp. 1–7.

[25] W. Guan, X. Wen, L. Wang, Z. Lu, and Y. Shen, "A service-oriented deployment policy of end-to-end network slicing based on complex network theory," IEEE Access, vol. 6, pp. 19 691–19 701, 2018.

[26] G. Sun, Z. T. Gebrekidan, G. O. Boateng, D. Ayepah-Mensah, and W. Jiang, "Dynamic reservation and deep reinforcement learning based autonomous resource slicing for virtualized radio access networks," IEEE Access, vol. 7, pp. 45 758–45 772, 2019.

[27] 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 Journal on Selected Areas in Communications, vol. 38, no. 6, pp. 1040–1057, 2020.

[28] G. Brown, "Service-based architecture for 5g core networks," white paper, Huawei Technologies Co. Ltd., Tech. Rep., 2017.

[29] W. Lei, A. C. K. Soong, and L. Jianghua, 5G Capability Outlook: ITU-R Submission and Performance Evaluation. Cham: Springer International Publishing, 2020, pp. 299–369. [Online]. Available: https://doi.org/10.1007/978-3-030-22236-9_5

[30] ITU-R, "Minimum requirements related to technical performance for imt-2020 radio interface(s)," 2020. [Online]. Available: https://www.itu.int/pub/R-REP-M.2410

[31] M. Wydmuch, M. Kempka, and W. Jaśkowski, "Vizdoom competitions: Playing doom from pixels," IEEE Transactions on Games, 2018.

[32] M. S. Munir, S. F. Abedin, N. H. Tran, Z. Han, E.-N. Huh, and C. S. Hong, "Risk-aware energy scheduling for edge computing with microgrid: A multi-agent deep reinforcement learning approach," IEEE Transactions on Network and Service Management, 2021.

[33] M. Alsenwi, N. H. Tran, M. Bennis, S. R. Pandey, A. K. Bairagi, and C. S. Hong, "Intelligent resource slicing for embb and urllc coexistence in 5g and beyond: A deep reinforcement learning based approach," IEEE Transactions on Wireless Communications, 2021.

[34] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," IEEE Communications Surveys Tutorials, vol. 15, no. 4, pp. 1888–1906, 2013.

[35] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: Substrate support for path splitting and migration," Computer Communication Review, vol. 38, pp. 17–29, 04 2008.

[36] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "22.2 breadth-first search," 2009.

[37] S. Wimmer and P. Lammich, "The floyd-warshall algorithm for shortest paths," Archive of Formal Proofs, May 2017, http://isa-afp.org/entries/Floyd_Warshall.html, Formal proof development.

[38] A.-L. Barabasi and R. Albert, "Albert, r.: Emergence of scaling in random networks. science 286, 509-512," Science (New York, N.Y.), vol. 286, pp. 509–12, 11 1999.

[39] R. E. Hattachi and J. Erfanian, "Ngmn 5g white paper," NGMN Alliance, Tech. Rep., 2015.

[40] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural computation, vol. 9, pp. 1735–80, 12 1997.

[41] S. Haykin, Neural Networks: A Comprehensive Foundation. Prentice Hall, 2nd Ed., 1999.

[42] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[43] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," IEEE Signal Processing Magazine, vol. 34, no. 6, pp. 26–38, 2017.