IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 17 March 2023, accepted 6 April 2023, date of publication 24 April 2023, date of current version 27 April 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3269576

RESEARCH ARTICLE

Deep Multi-Agent Reinforcement Learning With Minimal Cross-Agent Communication for SFC Partitioning

ANGELOS PENTELAS[1,2], (Student Member, IEEE), DANNY DE VLEESCHAUWER[1], (Member, IEEE), CHIA-YU CHANG[1], (Member, IEEE), KOEN DE SCHEPPER[1], (Associate Member, IEEE), AND PANAGIOTIS PAPADIMITRIOU[2], (Senior Member, IEEE)

[1]Nokia Bell Laboratories, 2018 Antwerp, Belgium
[2]Department of Applied Informatics, University of Macedonia, 546 36 Thessaloniki, Greece

Corresponding author: Angelos Pentelas (apentelas@uom.edu.gr)

This work was supported by the European Union's Horizon 2020 Research and Innovation Program under Grant 101017109 (DAEMON).

ABSTRACT Network Function Virtualization (NFV) decouples network functions from the underlying specialized devices, enabling network processing with higher flexibility and resource efficiency. This promotes the use of *virtual network functions* (VNFs), which can be grouped to form a *service function chain* (SFC). A critical challenge in NFV is SFC partitioning (SFCP), which is mathematically expressed as a *graph-to-graph* mapping problem. Given its NP-hardness, SFCP is commonly solved by approximation methods. Yet, the relevant literature exhibits a gradual shift towards data-driven SFCP frameworks, such as (deep) reinforcement learning (RL). In this article, we initially identify crucial limitations of existing RL-based SFCP approaches. In particular, we argue that most of them stem from the centralized implementation of RL schemes. Therefore, we devise a cooperative deep multi-agent reinforcement learning (DMARL) scheme for decentralized SFCP, which fosters the efficient communication of neighboring agents. Our simulation results (i) demonstrate that DMARL outperforms a state-of-the-art centralized *double deep Q-learning* algorithm, (ii) unfold the fundamental behaviors learned by the team of agents, (iii) highlight the importance of information exchange between agents, and (iv) showcase the implications stemming from various network topologies on the DMARL efficiency.

INDEX TERMS Multi-agent reinforcement learning, network function virtualization, self-learning orchestration.

I. INTRODUCTION

Increasing processing demands, typically coupled with highly stringent performance requirements, put modern networks under severe stress. To mitigate the risk of under-delivery, domain experts promptly re-conceptualized several architectural components of networks, and pushed innovations that render them more agile, cost-effective, and high-performing. One such technology is Network Function Virtualization (NFV), which comprises a paradigm shift from hardware-based to virtual network functions (VNFs). As such, common network functions can be implemented in the form of virtual machines or containers and be deployed on-demand within commercial off-the-shelf (COTS) servers. In addition, multiple VNFs are arranged into an ordered sequence to form a service function chain (SFC), which in effect enforces an end-to-end flow processing policy.

Nonetheless, instilling such a high degree of flexibility into networks via NFV comes with a certain cost, which in this case is embodied by *management* and *orchestration* (MANO) complexity. Indeed, instantiating, configuring, monitoring, and scaling SFCs comprise only a subset of operations that need to be applied upon these new network constructs. To support such features, the whole NFV endeavour is

The associate editor coordinating the review of this manuscript and approving it for publication was Abderrahmane Lakas.

backed by the NFV MANO framework, which consists in a hierarchical architecture that positions an NFV orchestration (NFVO) module on top of virtualized infrastructure managers (VIMs). Effectively, each VIM operates on a set of COTS servers (which are typically deployed in a datacenter) and has a detailed view on critical local parameters, *e.g.,* resource consumption levels, (anti-)affinity constraints, volume bindings, etc. Yet, useful information from multiple VIMs shall be conveyed to the NFVO layer, since certain actions of the latter might require the coordination of more than a single VIM. This is the case for the SFC partitioning (SFCP) problem, where the diverse location constraints of VNFs raises the need for their distribution across multiple VIMs.

From an algorithmic point of view, SFCP is proven to be an NP-hard optimization problem [1], which practically implies scalability limitations. As being typical in problems of such computational complexity, many studies model SFCP as a mixed-integer linear program (MILP), and subsequently propose heuristics or approximation algorithms that can cope with the scalability issue at the expense of solution quality (*e.g.,* [2], [3]). However, these algorithms are usually tailored to specific environments (*e.g.,* certain network topologies), constraints and objectives, thereby requiring drastic redesign in case some problem parameters or goals alter. Additionally, they generally assume perfect knowledge with regard to the state of the physical network, which is highly unrealistic due to high network dynamics.

To overcome these limitations, data-driven resource allocation techniques, primarily based on reinforcement learning (RL), have recently started to gain traction within the problem space of SFCP (*e.g.,* [4], [5], [6], [7], [8], [9], [10], [11], [12]). This algorithmic shift has its grounds on several arguments. First, rapid advancements in the RL field and, in particular, the application of deep neural networks within the RL framework [13], empowers it to handle vast *state* spaces. Prior to this development, RL methods were unable to cope with large environments since quantifying each *state-action* combination required testing it multiple times. Second, RL methods are inherently purposed to design their own optimization strategies (*i.e.,* policies). They achieve this merely by interacting with the environment upon which they are applied and by receiving a problem-specific reward signal that expresses the decision quality. Given that these features are aligned with the *zero-touch network automation* endeavour set out by operators, it is only natural that RL algorithms have become state-of-the-art SFCP methods.

A. LIMITATIONS OF CENTRALIZED SFCP WITH RL

With regard to centralized and RL-driven SFCP, the relevant literature exhibits certain limitations, the most crucial of which are listed and analyzed below:

1) ASSOCIATION OF INTENTS

A critical limitation of centralized SFCP with RL is the association of the resource allocation intents of the NFVO with the respective intents of the underlying VIMs. That is, as it is common in hierarchical resource management systems with global and local controllers (*e.g.,* NFVOs and VIMs, VIMs and hypervisors, k8s master and worker nodes), the global controller queries the local controllers about their available resources, and uses this information to compute a resource allocation decision (which is eventually realized by the latter). Effectively, this limits the inherent capacity of the local controllers to express their own resource allocation intents. In fact, global and local intents are not necessarily conflicting; on the contrary, fostering the acknowledgement of local intents can improve the resource allocation efficiency of the overall system, since each local controller has a more precise view of its actual state.

2) DECISION-MAKING WITH INCOMPLETE INFORMATION

The common assumption that a single learning agent, positioned at the NFVO layer (which is the standard practice in centralized SFCP with RL), has a precise view over the entire topology, is somehow unrealistic. In fact, the higher we move along the NFV MANO hierarchy, the more coarse-grained the information we have at our disposal (because of data aggregation), which implies higher uncertainty. Conversely, if a centralized RL approach admits partial observability, it follows that the decision-making process relies on incomplete information.

3) DEPENDENCE OF ACTION SPACE ON TOPOLOGY NODES

In most works that propose centralized RL schemes for the SFCP problem, the action space of the learning agent coincides with the set of available points-of-presence (PoPs). This is somewhat natural, since the centralized agent has to infer which is the best PoP to host a particular VNF. However, the dependence of the agent's architecture on the physical nodes of the substrate makes the RL scheme particularly hard to scale. That is, the larger the action set, the longer the training time. Our argument is further strengthened by studies which employ techniques for shrinking the action space, *e.g.,* clustering a set of points-of-presence (PoPs) into a single group, or placing the entire SFC within a single PoP (*e.g.,* [7], [11]).

B. CONTRIBUTIONS

Admittedly, targeting for a single solution that addresses all of the aforementioned limitations would be extremely optimistic. However, one can easily observe that most shortcomings stem from the centralized implementation of SFCP. Indeed, a decentralized approach that assigns learning agents locally to each VIM, in conjunction with a module at the NFVO layer that coordinates the resulting local decisions, would alleviate many of these limitations. In particular, (i) the global and local intents would be naturally decoupled, (ii) each agent would act based on detailed local observations; thus, from the perspective of the overall system, the true state would be fully observable, and (iii) it would be easier to decouple action spaces from the physical topology. Along

these lines, this work focuses on the investigation, the development, and the evaluation of a decentralized SFCP scheme based on cooperative deep multi-agent reinforcement learning (DMARL). Our main contributions are the following:

- We devise a DMARL framework for decentralized SFCP. Our DMARL algorithm consists of independent *double deep Q-learning* (DDQL) agents, and fosters the exchange of concise, yet critical, information among them.
- We develop a DDQL algorithm for centralized SFCP, and compare its performance with DMARL.
- We quantify fundamental rules identified by the team of agents over the course of training.
- We examine the impact of (imperfect) cross-agent communication across various substrate network topologies.

The remainder of this article is structured as follows. In Section II, we give a solid description of SFCP and basic formulations for system models. In Section III, we elaborate on deep RL and present a single-agent DDQL algorithm for SFCP. In Section IV, we cast SFCP as a multi-agent learning problem, and devise a DMARL algorithm to address it. Section V includes our evaluation results and in Section VI we discuss related work. A thorough discussion is given in Section VII, and conclusions are laid out in Section VIII.

II. PROBLEM FORMULATION

In this section, we commence with a high-level description of SFCP and, subsequently, we formalize important concepts via system models.

A. PROBLEM DESCRIPTION

As already explained, a SFC is an ordered sequence of VNFs, which enforces a flow processing policy. For instance, *Firewall* → *Packet inspection* → *Load Balancer* is a typical SFC which can be seen useful in, *e.g.,* securing and load balancing a web application. Effectively, flows have to traverse the entire set of VNFs with the specified order to reach their destination.

To delve deeper, each VNF of the SFC requires computing resources (*e.g.,* CPU) for packet processing. Additionally, connections among consecutive VNFs, henceforth termed as virtual links (*vlinks*), require network resources (*e.g.,* bandwidth) in order to forward traffic from one VNF to another. Such computing and network resources are offered by a multi-datacenter system. Specifically, computing resources are offered by servers, while network resources are allocated from physical links. It is also quite common to assume that computing resources are offered by a PoP (*e.g.,* a datacenter).

Whether PoPs belong to a single or multiple administrative domains has a crucial effect on SFCP. Concretely, in multi-domain settings (where PoPs are managed by multiple administrators), resource utilization parameters are confined within each domain. Thereby, SFCP is typically solved via resource bidding mechanisms, *e.g.,* [14], [15], and [16]. On the other hand, there are no information disclosure concerns in single-domain scenarios. However, this leads to the common misunderstanding that each bit of information can be easily conveyed to a centralized controller for decision-making with full observability. While theoretically possible, the above practice is simply ineffective, since it implies high communication overheads [17]. Effectively, it is more realistic to assume that PoPs expose descriptive statistics about their resources, such as the average residual CPU capacity of their servers. To this end, our work considers single-domain settings, and it accounts for the limited information that can be exchanged between PoPs and the centralized controller.

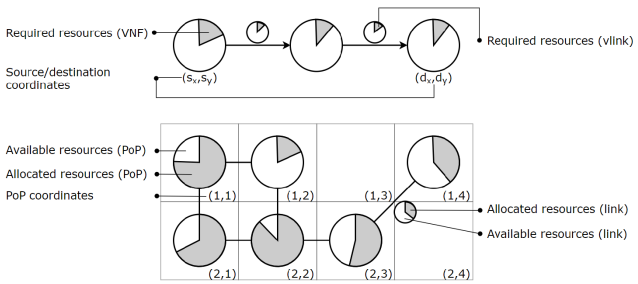


FIGURE 1. An SFCP problem instance. On top, an SFC-R G_R. At the bottom, a multi-PoP topology G_S. Resources of physical links are not exhaustively depicted for figure clarity.

Irrespective of the number of domain administrators, SFCP boils down to the problem of computing a mapping between the virtual elements of an SFC and their physical counterparts, while adhering to resource capacity constraints. Specifically, VNFs are assigned to PoPs in a one-to-one fashion, while virtual links are assigned to physical links in a one-to-many fashion. Naturally, the VIM of each PoP then needs to compute a placement of the sub-SFC elements (*i.e.,* subset of VNFs and vlinks) onto physical servers and intra-datacenter links, but this topic is not within the scope of our work. An illustration of an SFCP problem instance is given in Fig. 1. An SFC (consisting of three VNFs) at the top of the figure is considered to be partitioned over a six-PoP topology at the bottom of the same figure. Both VNFs and virtual links require resources (dark circular sectors), while PoPs and physical links are characterized by available and allocated resources (light and dark circular sectors, respectively). Further, each SFC contains a source and a destination, while each PoP is represented by its coordinates in a two-dimensional space.

Optimality with respect to SFCP can take numerous forms. For instance, it can be seen through the lens of load balancing [18], throughput maximization [5], latency minimization [3], fault tolerance (reliability) [11], cross-service communication [19], or resource efficiency [20]. Our work does not intend neither to add another problem dimension nor to further explore an existing one; it rather focuses on the investigation of an RL scheme that is more aligned with current system architectures and goals. Here, we opt for the latency minimization objective, since a key problem for SFCP is to fulfill the stringent low-latency requirements.

B. SYSTEM MODELS

1) SERVICE FUNCTION CHAIN REQUEST (SFC-R) MODEL

We model an SFC-R with the 5-tuple $< G_R, src, dst, t, \Delta t >$. $G_R = (V_R, E_R)$ is a directed graph, where V_R and E_R denote the set of nodes and edges (*i.e.,* VNFs and virtual links), respectively. Additionally, $d_R(i)$ expresses the CPU demand of VNF $i \in V_R$, while $d_R(i,j)$ represents the bandwidth requirements of $(i,j) \in E_R$ (both expressed as percentages). *src* and *dst* indicate the source and the destination coordinates of the SFC, while t expresses its arrival time, and Δt denotes its lifespan.

We note that both *src* and *dst* are modeled as auxiliary VNF nodes with zero resource requirements (*i.e., src*, *dst* $\in V_R$, and $d_R(src) = d_R(dst) = 0$). Naturally, this introduces two additional edges to the SFC graph, one from the *src* towards the first VNF, and one from the last VNF towards the *dst* (see top of Fig. 3). Further, we assume that the *src* and *dst* of an SFC correspond to coordinates of arbitrary PoPs, denoted by u_{src} and $u_{dst} \in V_S$.

2) SUBSTRATE NETWORK MODEL

We model a substrate network with an undirected graph $G_S = (V_S, E_S)$, where V_S and E_S denote the sets of nodes and edges. Since we consider G_S to be a PoP-level topology (*i.e.,* a network of datacenters), V_S corresponds to PoPs, and E_S to inter-PoP links. $d_{S,t}(u)$ expresses the available CPU of node u at time t (*i.e.,* the average available CPU of servers belonging to PoP u), and $d_{S,t}(u,v)$ denotes the available bandwidth of the edge (u,v) at time t (again, as percentages). Last, $D(u,v)$ represents the propagation delay of a physical link (u,v).

III. SINGLE-AGENT REINFORCEMENT LEARNING

This section discusses single-agent RL optimization. We initially outline the common RL setting, and proceed with a short description of the state-of-the-art DDQL algorithm. Subsequently, we elaborate on our proposed DDQL approach for SFCP.

A. BACKGROUND

The typical RL setting considers a learning-agent, an environment, a control task subject to optimization, and a discrete time horizon $H = \{1, 2, \ldots\}$, which can as well be infinite. At time $t \in H$, the agent observes the current state of the environment $s_t \in S$, with S indicating the entire set of possible states. Equipped with a set of actions X, the agent shall interact with the environment by choosing $x_t \in X$. Then, the agent receives feedback regarding the quality of its action via a reward signal r_t, and the environment transitions to the subsequent state $s_{t+1} \in S$.

Conventionally, a RL task is modelled as a Markov decision process, defined by the tuple $< S, X, S_1, T, R >$. Here, S and X are as before. $S_1 \in P(S)$ denotes the starting state distribution, and $T : S \times X \rightarrow P(S)$ is the state transition function, where $P(\cdot)$ denotes the set of probability distributions over a set. Finally, $R : S \times X \times S \rightarrow \mathbb{R}$ expresses the reward function, which is generally denoted as R_t instead of $R(s_t, x_t, s_{t+1})$ for simplicity. Naturally, the aim of the agent is to maximize the accumulated discounted reward over H, *i.e.,* $\sum_{t \in H} \gamma^t R_t$, where $\gamma \in [0, 1)$ is a discount factor employed to prioritize immediate reward signals against those projected farther into the future. To achieve that, the agent needs to perform a sequence of actions based on a learned *policy* π, which is practically a function that maps states to probability distributions over actions, *i.e.,* $\pi : S \rightarrow P(X)$, and the optimal policy is denoted as π^*.

An important concept here is the *action-value function*, which quantifies the agent's incentive of performing a specific action on a particular state:

$$Q(s_t, x_t) := \mathbb{E}_t \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} | s_t, x_t \right], (s_t, x_t) \in S \times X \quad (1)$$

Apparently, if the agent explores its action selection on the environment long enough such that Q-values represent ground truth reward values, then finding the optimal policy becomes trivial. That is, $\pi^*(s_t) = argmax_{x \in X} Q(s_t, x)$. However, Q-values are treated as estimates, since the assumption of perfect exploration is hardly ever realistic. To this end, the agent needs to balance *exploration* and *exploitation*, where the former shall strengthen its confidence on Q estimations, and the latter will enable it to benefit from accumulated knowledge. The most common approach to achieve a favourable exploration-exploitation trade-off is via an $\epsilon - greedy$ action selection strategy. According to it, the agent performs a random action with probability ϵ, while with probability $1 - \epsilon$ the action with the highest Q-value is selected. Given an initial ϵ, *i.e.,* ϵ_0, and a decay factor $\epsilon_{decay} \in (0, 1)$, the agent can gradually shift from exploratory to exploitative:

$$\epsilon_t = \epsilon_0 \cdot (\epsilon_{decay})^t \quad (2)$$

B. DOUBLE DEEP Q-LEARNING

A well-known method that leverages the notions above is *Q-learning*, which is a *model-free*,[1] *off-policy*[2] RL algorithm. Q-learning attempts to find optimal policies via a direct estimation of Q-values, which are maintained in a tabular format. Each time a state-action pair (s_t, x_t) is visited, the respective Q-value is updated. An apparent restriction here is the lack of generalization capacity. That is, in problems with large state-action spaces, the risk of scarce Q-updates is high.

To overcome this limitation, Mnih et al. [13] propose a Q-function approximation scheme based on deep neural networks (NNs), commonly termed as deep Q-learning (DQL). As per DQL, the learning capacity of the agent lies in a NN which approximates the Q-function by parameterizing it, *i.e.,* $Q(s, x) \sim Q(s, x; \theta)$ (where θ represents a vector of weights and biases of the NN). Ultimately, given a state as input, the

[1] It does not intend to discover the transition function.

[2] While following a specific policy, it assesses the quality of a different one.

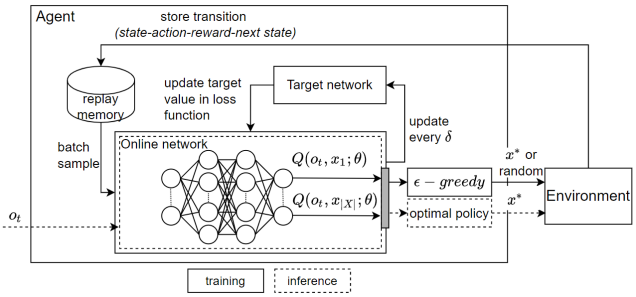


FIGURE 2. Illustration of the typical DDQL architecture.

NN computes a Q-value for every possible action. Then, it is up to an *action selection strategy* (*e.g.,* $\epsilon - greedy$) to choose a particular action.

Two mechanisms that contribute to the success of DQL are the *replay memory* and the coordination of an *online* NN and a *target* NN [21]. The replay memory is used to store *state-action-reward-next state* transitions; with a proper sampling over it, we can subsequently train the online NN with uncorrelated data. The interplay of the two NNs works as follows: initially, the target NN is an exact replica of the online NN, meaning that they share the same architecture, weights and biases. However, it is only the online NN that is updated at every training step, while its weights and biases are copied to the target NN every δ training steps. The target NN (represented by θ^-) is used to temporarily stabilize the target value which the online NN (represented by θ) tries to predict. Effectively, the training of the learning agent boils down to the minimization of Eq. (3), also known as *loss function*. In the *target* term of Eq. (3), the greedy policy is evaluated by the online NN, whereas the greedy policy's value is evaluated by the target NN. As per [21], this reduces the overestimation of Q-values, which would be the case if both the greedy policy and its value have been evaluated by the online NN. An illustration of the DDQL framework is given in Fig. 2.

$$L(\theta_t) = \mathbb{E}\Big[\underbrace{\Big(r_t + \gamma Q\big(s_{t+1}, \underset{x \in X}{argmax} Q(s_{t+1}, x; \theta_t); \theta_t^-\big)}_{\text{target}} - \underbrace{Q(s_t, x_t; \theta_t)}_{\text{predicted}}\Big)^2\Big] \tag{3}$$

C. DDQL FOR SFCP

We devise a single-agent RL algorithm for SFCP to serve as a baseline for comparison with our multi-agent RL scheme. Specifically, we opt for the DDQL method, as it has demonstrated promising results within our problem space (*e.g.,* [7], [11], [16]). In our implementation, a training episode consists of the placement of an entire SFC, whereas a decision step refers to the placement of a single VNF of the SFC.

1) OBSERVATIONS

At time t, the agent observes information o_t about the current VNF i and the SFC G_R that it belongs to, as well as information about the available resources of the substrate physical topology G_S. In particular, the agent receives the length of the SFC ($|V_R|$), the coordinates of the *src* (*src.loc*) and *dst* (*dst.loc*) nodes of the SFC, the CPU demand of VNF i ($d_R(i)$), the ingress ($d_R(i^-, i)$) and egress ($d_R(i, i^+)$) bandwidth requirements of VNF i, as well as its order (*i.order*) in the SFC. Regarding the physical network, the agent observes the coordinates (*u.loc*) and the average available CPU ($d_{S,t}(u)$) of each PoP $u \in V_S$, and the available bandwidth ($d_{S,t}(u, v)$) of each link $(u, v) \in E_S$. That is:

$$o_t = \Big(\underbrace{|V_R|, src.loc, dst.loc}_{\text{SFC state}}, \underbrace{d_R(i), d_R(i^-, i), d_R(i, i^+), i.order}_{\text{VNF state}}, \underbrace{(u.loc, d_{S,t}(u), \forall u \in V_S), (d_{S,t}(u, v), \forall (u, v) \in E_S)}_{\text{substrate state}}\Big)$$

Notice that, since o_t does not hold information about the individual servers of the topology (which are the elements that actually host VNFs), the environment is partially observable, hence we use *observation* o_t instead of *state* s_t.

2) ACTIONS

In our centralized implementation, an action determines which PoP will host the current VNF i. Concretely, the agent's action set is $X = \{1, 2, \ldots, |V_S|\}$. Notice that, if decision steps referred to the placement of an entire SFC instead of individual VNFs, then the action space would have been substantially larger (*i.e.,* $|X| = |V_S|^{|V_R|}$). This would have an adverse effect on the algorithm's performance.

3) REWARD

A VNF placement is deemed successful, if the selected PoP has at least one server with adequate resources to host it. An SFC placement (which is the ultimate goal) is deemed successful, if all of its VNFs have been successfully placed and all of its virtual links are assigned onto physical paths that connect the VNFs correctly. For every successful VNF placement, the agent receives $r_t = 0.1$. If the current VNF is the last VNF of the chain (*i.e., terminal* VNF) and is successfully placed, then the virtual link placement commences (using Dijkstra's shortest path algorithm for every adjacent VNF pair - similar to [6] and [10]). If this process is successfully completed, the reward is computed as follows:

$$r_t = 10 \cdot \frac{|opt\_path| + 1}{|act\_path| + 1} \tag{4}$$

where $|opt\_path|$ is the length of the shortest path between the *src.loc* and the *dst.loc* (recall that these are always associated with PoP locations), and $|act\_path|$ is the length of the actual path established by the DDQL algorithm. Apparently, the best reward $r_t = 10$ is given when $|opt\_path| = |act\_path|$.

Algorithm 1 DDQL Training Procedure

```
1:  Input: sfc_dataset, topology
2:  env = Environment(sfc_dataset, topology)
3:  agent = DDQLAgent(topology)
4:  for sfc in sfc_dataset do {training episode}
5:      done = False
6:      state = env.reset()
7:      env.place_src_dst()
8:      score = 0
9:      while not done do {training step}
10:         action = agent.choose_action(state)
11:         new_state, reward, done = env.step(action)
12:         score += reward
13:         agent.store(state, action, reward, new_state)
14:         agent.learn() {train the online NN}
15:         state = new_state
16:     end while
17: end for
```

Algorithm 2 Environment Transition Procedure

```
1:  Input: action
2:  done = False
3:  PoP = decode(action)
4:  n_success = place_node(vnf, PoP)
5:  if n_success and vnf.is_terminal then
6:      l_success, act_path, opt_path = place_links(sfc, topo)
7:      if l_success then {successful SFCP}
8:          reward = 10 · (|opt_path|+1)/(|act_path|+1)
9:      else {insufficient link capacities}
10:         reward = −10
11:     end if
12:     new_state = next_sfc()
13:     done = True
14: else if n_success then {successful VNF allocation}
15:     reward = 0.1
16:     new_state = next_vnf()
17: else {insufficient PoP capacity}
18:     reward = −10
19:     new_state = next_sfc()
20:     done = True
21: end if
22: return new_state, reward, done
```

In case the placement of any VNF or virtual link fails (due to inadequate physical resources), then $r_t = -10$ and a new training episode initiates.

4) ARCHITECTURE

As hinted by previous discussions, the functionality of our DDQL agent relies on four key elements, namely an online NN, a target NN, a replay memory, and an action selection strategy. Both NNs comprise an input layer whose size equals the length of the observation vector o_t (*i.e.,* $|o_t| = 3|V_S| + |E_S|+9$), two fully-connected hidden layers with 256 neurons each, and an output layer with $|V_S|$ neurons. All layers are feed-forward, the activation function applied on individual cells is Rectified Linear Unit (ReLU), and the loss function used is Eq. (3). The replay memory is implemented as a queue, which stores the latest 10, 000 environment transitions. The online NN samples 64 random transition instances out of the replay memory at every training step, while the target NN is updated every $\delta = 20$ steps. Last, we employ an $\epsilon - greedy$ action selection strategy, where, in Eq. (2), we set $\epsilon_0 = 1$ and $\epsilon_{decay} = 0.9998$.

5) TRAINING

The instrumentation of the above is summarized in Algs. 1 and 2. In more detail, Alg. 1 describes the steps of the DDQL training process, which takes place over a collection of SFCs (*sfc_dataset*) and a physical PoP topology (*topology*) (**line 1**). At each episode (**line 4**), the agent handles the placement of a unique SFC. Prior to any action, the environment resets to a new state in **line 6** (*i.e.,* we increment the SFC index in the *sfc_dataset* by one, obtain the first VNF of the current SFC, and generate random loads for PoPs and physical links), and assigns the *src* and *dst* of the SFC to the respective PoPs (**line 7**). The core learning process lies within **lines 9-16**. First, the agent chooses an action based on the current state (**line 10**), the environment transitions to a new state based on the action being taken (**line 11**), the transition is stored in the replay memory of the agent (**line 13**), the online NN is trained (**line 14**), and the current score and state are updated (**lines 12** and **15**). The transitions of the environment to a new state are further described in Alg. 2, which practically implements the reward computation and episode termination logic described earlier (see the *Reward* paragraph).

IV. MULTI-AGENT REINFORCEMENT LEARNING

We herein introduce cooperative multi-agent reinforcement learning (MARL). Then, we discuss independent Q-learning, which is the foundation of the proposed solution. Finally, we describe in-detail a DMARL scheme for SFCP.

A. COOPERATIVE MARL

MARL builds upon the fundamental blocks of single-agent RL, presented in Section III-A. In particular, MARL considers a set of n agents $A = \{1, \ldots, n\}$ interacting with the environment. At each time step $t \in H$, each agent $\alpha \in A$ observes o_t^α, and draws an action x_t^α from its own designated action set X^α. The joint action $\boldsymbol{x_t} = (x_t^1, \ldots, x_t^n)$ is then applied on the environment, which transitions to a new state s_{t+1}. In extension, each agent α observes o_{t+1}^α. Similar to single-agent RL, the transition $(s_t, \boldsymbol{x_t}, s_{t+1})$ is evaluated by means of a reward function R, and a scalar r_t^α is sent to each agent. In a fully cooperative MARL setting, all agents share the same reward, *i.e.,* $r_t^\alpha = r_t,\ \forall\, \alpha \in A$.

In principle, the dynamics of the above scheme can be captured by a *cooperative Markov game*, typically defined

by the tuple $< S, \boldsymbol{X}, S_1, T, R, Z, O, n >$. Here, S, S_1, T and R are as in Section III-A. $\boldsymbol{X} = X^1 \times \ldots \times X^n$ denotes the joint action space, Z expresses the space of observations, and $O : S \times A \rightarrow Z$ is the observation function that dictates the partial observability of each agent (*i.e.,* O can be seen as the function that maps a state s_t and an agent α to an observation $o_t^\alpha \in Z$). In this setting, the goal of each agent α is to discover a policy $\pi^\alpha : Z \rightarrow P(X^\alpha)$, such that the joint policy[3] $\boldsymbol{\pi} = (\pi^1, \ldots, \pi^n) : S \rightarrow P(\boldsymbol{X})$ maximizes the (discounted) accumulated reward.

B. INDEPENDENT Q-LEARNING

Undoubtedly, the simplest way of establishing a cooperative MARL framework is to treat each learning agent as an independent learning module. In this setting, if each agent is realized as a Q-learning agent, the respective MARL system is known as *independent Q-learning* (IQL) [22]. A major limitation of IQL is its lack of convergence guarantees, primarily stemming from the fact that the environment is non-stationary from the perspective of each agent. Effectively, this means that for an agent α, both its reward r_t and its next observation o_{t+1}^α are not solely conditioned on its current observation o_t^α and action x_t^α. In other words, the transitions of the environment, and, in extension, the common rewards, are affected by the actions of other agents as well. Even though obtaining an optimized joint policy $\boldsymbol{\pi}$ while agents treat other agents as part of the environment is seemingly difficult, IQL exhibits good empirical performance [23].

Cooperative MARL is an active research field, given that numerous control tasks can be seen through the lens of multi-agent systems. In the context of collaborative DQL agents, recent works (*e.g.,* [24] and [25]) have established frameworks that promote joint training of multiple agents (*i.e., centralized training - decentralized execution*), under the assumption that the Q^{total}-function of the multi-agent system can be decomposed into individual Q-functions such that, if each agent α maximizes its own Q^α, then Q^{total} is also maximized.

Irrespective of the promising advances in the field, our work builds upon typical IQL for the following reasons. First, IQL is simple enough to facilitate the interpretation of the system's learning behavior, as it avoids complex NN training schemes. Second, as it is apparent in Section IV-C, we envisage an action *coordination* module at the NFVO layer to handle the constraints of SFCP, which can significantly benefit the overall IQL framework.

C. DMARL FOR SFCP

We implement a DMARL scheme for SFCP. Specifically, we generate a single DDQL agent (as described in Section III-B) for every PoP u in the substrate network G_S. For the rest of this section, we assume that agent α is associated with PoP u, and agent β with PoP v.

[3]Here, we implicitly assume that the union of partial observations of all agents can compose the state space, *i.e.,* $\cup_{\alpha \in A} o_t^\alpha = s_t, \forall t$.

1) OBSERVATIONS

At time t, each agent $\alpha \in A$ observes information o_t^α about the current VNF i and the SFC G_R to which this VNF belongs, as well as the available resources of the substrate physical topology G_S. In detail, agent α receives the length of the SFC, the coordinates of *src* and *dst* nodes of the SFC, the CPU demand of VNF i, the ingress and egress bandwidth requirements of VNF i, and the order of this VNF in the SFC, similar to the single-agent observation mentioned in Section III-C. Regarding the physical network, agent α observes the coordinates of PoP u, the available CPU ($d_{S,t}(u_s)$) of every server $s \in u$, and the available bandwidth of each physical link connected to PoP u.

We further augment the observation space of agents by enabling *cross-agent communications*. Specifically, we define $N(\alpha)$ to be the set of neighboring agents of α, *i.e.,* $N(\alpha) = \{\beta : distance(\alpha, \beta) = 1, \forall \beta \in A - \{\alpha\}\}$, where $distance(\alpha, \beta)$ refers to the length of the shortest path between PoPs u and v in G_S (recall that α operates over u and β over v). Effectively, α receives the location coordinates and the average CPU capacity from every PoP v managed by a neighboring agent $\beta \in N(\alpha)$. As it will become apparent, cross-agent communications are pivotal for the efficiency of the system, since they enable agents to reason about the state of other agents. Yet, to maintain the communication overhead low, agents do not share their entire local state (*i.e.,* the available CPU of each server - $d_{S,t}(u_s), \forall s \in u$), rather a single descriptive value (*i.e.,* the average available CPU across all servers - $d_{S,t}(u)$).

As explained in Section IV-B, agents within MARL settings operate over non-stationary environments, which practically implies that old experiences might become highly irrelevant as agents shift from exploratory to exploitative. To this end, we include a *fingerprint* [26] into the observation, namely ϵ (the probability to select a random action), as a means to distinguish old from recent experiences.

The last elements that are inserted into o_t^α are three binary flags, namely *hosts_another*, *hosts_previous*, and *in_shortest_path*, which are computed by agent α prior to action selection. In particular, *hosts_another* $= 1$ if any VNF of the current SFC has been placed in PoP u, *hosts_previous* $= 1$ if the previous VNF of the current VNF has been placed in PoP u, and *in_shortest_path* $= 1$ if PoP u is part of the shortest path between the *src* and the *dst* of the SFC, while they are zero otherwise. These three flags will be proven crucial for analyzing the behavior of the DMARL system in Section V-C. As such, we define:

$$o_t^\alpha = \big(\underbrace{|V_R|, src.loc, dst.loc}_{\text{SFC state}}, \underbrace{d_R(i), d_R(i^-, i), d_R(i, i^+), i.order}_{\text{VNF state}}, \underbrace{u.loc, (d_{S,t}(u_s), \forall s \in u), (d_{S,t}(u, v), \forall v \in N(u)), \epsilon}_{\text{local state (resources and fingerprint)}},$$

$$\underbrace{hosts\_another, hosts\_previous, in\_shortest\_path,}_{\text{local state (flags)}}$$
$$\underbrace{(v.loc, d_{S,t}(v), \forall v \in N(u))}_{\text{neighbors' condensed state}}$$

where $N(u)$ denotes the neighboring nodes of u in the graph G_S. Note that, contrary to the single-agent case, here the team of agents observes the true state of the environment, *i.e.,* $\cup_{\alpha \in A} o_t^\alpha = s_t$. However, from the point of view of a single agent, the environment is still partially observable.

2) ACTIONS

Agents are equipped with a set of actions that allows them to express their intents based on their local state. In detail, all agents share the same discrete set of actions $X = \{0, 1, 2\}$, where 0 indicates low willingness, 1 expresses a neutral position, and 2 indicates high willingness, with respect to hosting the current VNF. Note that the above action sets are topology-agnostic (in contrast to the action set of our single-agent DDQL, where there is one action for each PoP).

3) ACTION COORDINATION

The joint action $\boldsymbol{x_t} = (x_t^1, \ldots, x_t^n)$ of all agents is forwarded to an *action coordination* module positioned at the NFVO layer. This treats individual actions as *soft* decisions and eventually computes the *firm* decision according to the overall objective (*e.g.,* load balancing, consolidation) and constraints (*e.g.,* each VNF at exactly one PoP). Here, we opt for a simple action coordination scheme where (i) the agent with the highest action (ties broken arbitrarily) will enable its associated PoP to be selected to host the current VNF, and (ii) the objective is solely dictated by the reward function, meaning that the coordinator does not intend to achieve another objective. For example, if the joint action is $\boldsymbol{x_t} = (x_t^\alpha = 2, x_t^\beta = 0)$ for the placement of a VNF i, the action coordination module will infer that i will be positioned at node u, which is the PoP of agent α, since $x_t^\alpha > x_t^\beta$.

4) REWARD

Our DMARL algorithm adopts a reward scheme similar to the one presented in the single-agent case. That is, for every successful VNF placement, all agents receive $r = 0.1$, and for every successful SFC placement the common reward is computed in Eq. (4). In case of a failed SFCP attempt (inadequate physical resources), agents receive $r = -10$.

5) ARCHITECTURE

The proposed DMARL scheme is shown in Fig. 3, which illustrates the mapping of the proposed elements and functionalities onto the NFV MANO framework. At the top, we depict an SFC request which is conveyed to the NFVO via its northbound interface (NBI) in **step 0**. During **step 1**, the NFVO forwards the VNF and the SFC states to the underlying VIMs, and we assume that each VIM is associated with a single PoP. Additionally, each VIM is equipped with a DDQL

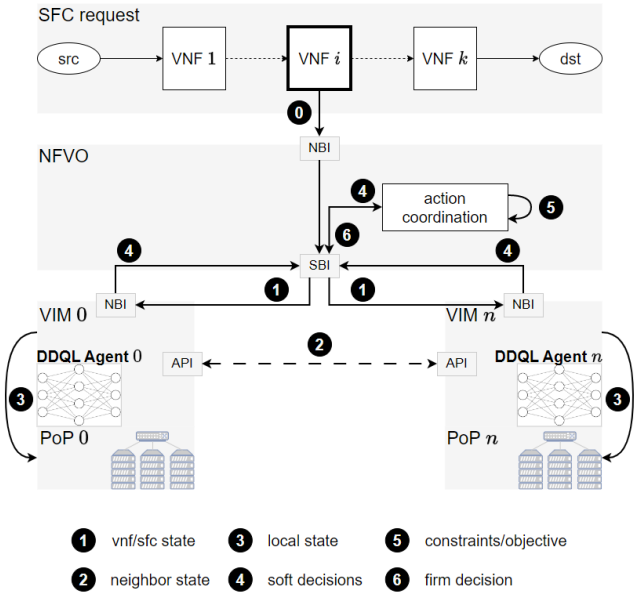


FIGURE 3. The proposed DMARL scheme is decomposed from step 0 to step 6 and mapped to the NFV MANO framework.

agent similar to the one described in Section III-C, their only difference being the input and output layers. In particular, the input layer of each DDQL agent α in the DMARL setting has $|o_t^\alpha|$ neurons, while its output layer consists of three neurons (one for each action). The API calls depicted in **step 2** implement the cross-agent communication functionality, where each agent retrieves condensed information from its neighbors. **Step 3** refers to the computation of local state, *i.e.,* monitoring of available CPU of each server and updating the three binary flag values. Afterwards, agents convey their soft actions via the NBIs of their VIMs towards the southbound interface (SBI) of the NFVO (**step 4**), where individual actions are aggregated into a joint action that is handed over to the coordination module. The latter resolves potential conflicts and, in principle, assesses the individual preferences with respect to the overall objective (**step 5**). Finally, the firm decision, which indicates the PoP that will host the current VNF, is sent to the respective VIM (**step 6**). This process is repeated until all VNFs and virtual links are assigned, or until the SFCP fails (virtual links are assigned via Dijkstra's method, similar to the single-agent case - cf. Section III-C).

6) TRAINING

The training procedure followed by the proposed DMARL scheme is outlined in Alg. 3, which exhibits many similarities with Alg. 1. Their core differences are as follows. First, in **line 1**, the number of actions (*n_actions*) is provided as input. That is, the number of actions is no longer determined by the number of PoPs; it rather becomes a parameter of the multi-agent framework. Specifically, the number of actions affects (i) the capacity of agents to express their resource allocation intents and (ii) the duration of the system's

Algorithm 3 DMARL Training Procedure

```
1:  Input: sfc_dataset, topology, n_actions
2:  env = Environment(sfc_dataset, topology)
3:  marl = MARL(topology, n_actions)
4:  for sfc in sfc_dataset do {training episode}
5:      done = False
6:      state = env.reset()
7:      env.place_src_dst()
8:      score = 0
9:      while not done do {training step}
10:         soft_actions = marl.choose_actions(state)
11:         firm_action = env.coordinate(soft_actions)
12:         new_state, reward, done = env.step(firm_action)
13:         score += reward
14:         marl.store(state, soft_actions, reward, new_state)
15:         marl.train() {train the online NNs of all agents}
16:         state = new_state
17:     end while
18: end for
```

convergence. As a good compromise, we always set the number of actions to three in this work, *i.e.,* actions are taken from the set $\{0, 1, 2\}$. Another difference between the two algorithms lies in the manner in which the *state* is interpreted in **line 6**. In Alg. 3, *state* refers to the VNF and the SFC state only, which is shared across all agents (see **step 1** in Fig. 3). However, in **line 10**, further subroutines are called so that each agent chooses an action based on additional observations, such as its local state and neighboring (condensed) states (cf. **steps 2** and **3** in Fig. 3). In **lines 11** and **12**, the coordination module derives the firm action, and the environment transitions to a new state based on this action, as explained in **steps 4** to **6** in Fig. 3. In **line 14**, each agent stores its transition into its replay memory (each agent expands the *state* variable), and in **line 15** all agents train their online NNs.

V. PERFORMANCE EVALUATION

This section covers a wide range of evaluation aspects, primarily focusing on the proposed DMARL algorithm. Initially, we present the simulation settings and the main performance metrics that we take into consideration. Then, we compare the single-agent DDQL method with the DMARL scheme, and elaborate on the learning behavior developed by the team of agents. Subsequently, we evaluate the cross-agent communication feature, and measure its implications across various physical network topologies. Finally, we experiment with an imperfect cross-agent communication scheme, which is more grounded to reality.

A. EVALUATION ENVIRONMENT AND PERFORMANCE METRICS

We consider linear SFCs consisting of two to five VNFs (excluding the *src* and *dst* auxiliary VNFs). Unless otherwise specified, the algorithms are trained with datasets containing 10, 000 SFCs (*i.e.,* 10, 000 training episodes). Each VNF of an SFC requires $5 - 20\%$ of a server's CPU. Each time a new SFC requests partitioning, the physical resources reset to arbitrary states. In particular, all previously embedded VNFs and virtual links are removed, and all servers generate random CPU loads within $70 - 100\%$. That is, their available CPU lies in $0 - 30\%$, and, in this way, we reduce the risk of precluding feasible partitionings due to CPU insufficiency. This approach (i) renders consecutive SFC placements (*i.e.*, episodes) independent events, and (ii) enables us to reason about the actual learning efficiency of the algorithms, as low scores will be solely due to insufficient learning. Further, each PoP comprises ten servers, *i.e.,* each PoP is a (micro) datacenter. Regarding the bandwidth of inter-PoP physical links, we assume that it always suffices for virtual link allocation in our current study, and aim to explore extensive insights into the aspect of virtual node allocations.

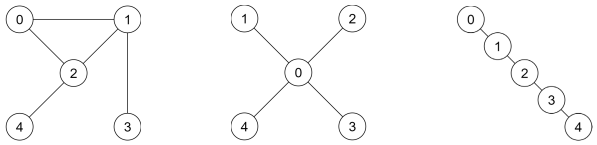


FIGURE 4. Topologies considered for evaluation.

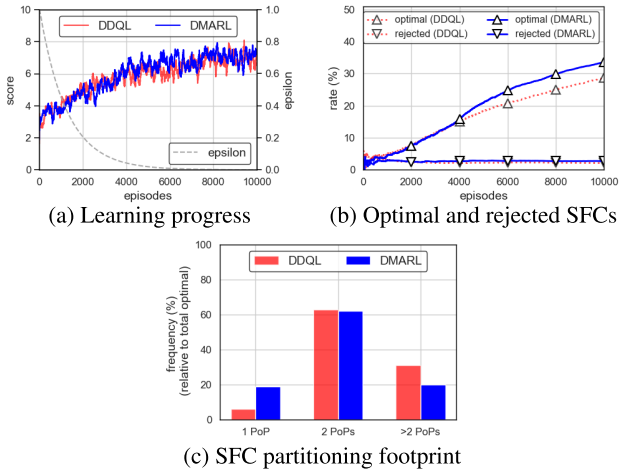
As illustrated in Fig. 4, we utilize three PoP topologies (*i.e., G_S*) in our evaluations, which have been selected to unveil critical properties of the multi-agent framework. In particular, each topology comprises five PoPs; hence, the (single-agent) DDQL algorithm works with five actions (one per PoP), and the DMARL framework generates five independent DDQL agents (one per PoP). We note that the auxiliary *src* and *dst* VNFs of an SFC are associated with random PoPs (*i.e., src.loc* and *dst.loc* might even coincide; in this case, the length of the optimal path $|opt\_path|$ is zero).

While we employ several micro-benchmarks in order to interpret and assess the algorithmic behaviors, two informative metrics refer to the tracking of (i) the accumulated reward, and (ii) the rate of optimal and rejected partitionings.

B. COMPARISON WITH STATE-OF-THE-ART

We compare the performance of the (centralized) DDQL against the (distributed) DMARL algorithm over the mesh topology shown in Fig. 4a. Recall that DDQL serves as our state-of-the-art benchmark method. Instead of using existing DDQL schemes for SFCP (*e.g.,* [7], [11], [16]), we devise a DDQL implementation (cf. Section III-C) which better matches our problem formulation and objective, thus allowing for a fair comparison.

At first glance at Fig. 5a, where we plot the corresponding scores as moving averages across the last 100 episodes, we observe that both methods perform similarly. In particular, towards the end of the experiment, the two algorithms manage to retrieve an average score of 7.0. Taking Eq. (4) into account, we infer that both learning schemes obtain SFCPs that are, on average, 30% from the optimal. However,



(a) Learning progress (b) Optimal and rejected SFCs

(c) SFC partitioning footprint

FIGURE 5. Performance comparison between DDQL and DMARL.

by zooming in at the $4,000 - 6,000$ episodes interval, it becomes apparent that the DMARL framework exhibits faster convergence towards the aforementioned score than DDQL. In fact, this is further corroborated by Fig. 5b, where it becomes clear that, at the exact same interval, the rate of optimal partitionings of DMARL grows more rapidly than that of DDQL. According to Fig. 5b, the rejection rates are negligible for both algorithms.

Although informative, Figs. 5a and 5b do not convey a lot about the differences of the underlying algorithmic behaviors. To this end, we also consider Fig. 5c, which depicts the footprint of optimal SFCPs. The most evident difference here is centered on the incapacity of DDQL to achieve many *1 PoP* partitionings, which is counterbalanced by its higher capacity to achieve *>2 PoP* partitionings, compared to DMARL. Our interpretation of this behavior is as follows: DDQL has a partial view of the real state, but its observation contains information about the entire topology (see Section III-C). This enhances its inherent capacity to distribute VNFs across many PoPs, since it can reason about the state of the entire multi-datacenter system. At the same time, its observations only include the average available CPU capacities of each PoP (*i.e.,* $d_{S,t}(u)$ in o_t), which limits its confidence in consolidating multiple VNFs into the same PoP. For instance, if $d_{S,t}(u) = 0.15$ for the PoP $u \in V_S$, and the current VNF i demands $d_R(i) = 0.18$, then it is impossible for DDQL to be certain whether i fits into a server of PoP u. Conversely, a DMARL agent α operating over u has a detailed view of the available CPU resources of all servers in PoP u, *e.g.,* α observes $(d_{S,t}(u_1) = 0.10, \ldots, d_{S,t}(u_s) = 0.20, \ldots, d_{S,t}(u_{10}) = 0.15)$, and, in this case, it can be certain that server s of PoP u can accommodate VNF i.

Although the aforementioned limitation can be mitigated by extending DDQL's observation space with additional metrics (at the risk of further slowing its convergence), we deem that this analysis indicates the implications of decision-making with incomplete information in the context of SFCP.

C. DMARL ANALYSIS

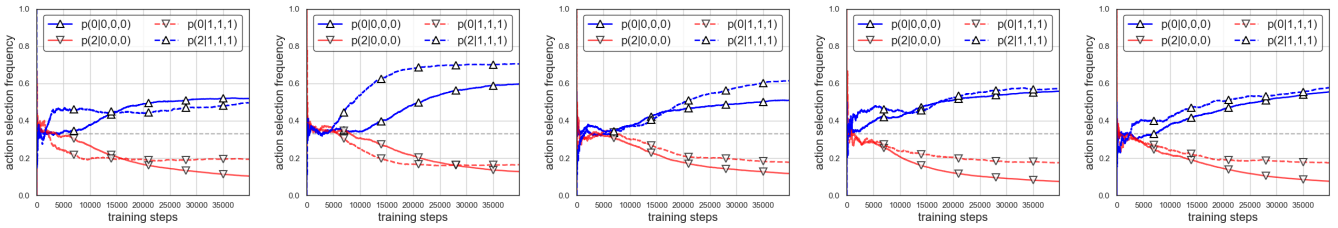
To further delve into the behavior developed by the DMARL scheme, we employ Fig. 6. Here, we attempt to shed light into the *learning* aspect of the team of agents in the mesh topology, by examining certain action selection *rules* that have been identified over the course of training. To this end, we monitor the evolution of $p(x|f_1, f_2, f_3)$ at every training step, which shall be interpreted as the probability of taking action $x \in X = \{0, 1, 2\}$, given that $f_1 = hosts\_another$, $f_2 = hosts\_previous$, and $f_3 = in\_shortest\_path$ (recall that these are binary flags, defined in Section IV-C, and are part of the observation o_t^α).

As per Figs. 6a - 6e, every agent manages to discover four key rules which, from a human perspective, seem rather intuitive for the SFCP problem. Specifically, the increase of $p(0|0, 0, 0)$ implies that the agents tend to express low willingness for hosting the current VNF when all flags are down, *i.e.,* no other VNFs are placed in the associated PoP ($hosts\_another = 0$), the previous VNF is placed in another PoP ($hosts\_previous = 0$), and the associated PoP is not in the shortest path from the *src* and the *dst* ($in\_shortest\_path = 0$). In contrast, a growing $p(2|1, 1, 1)$ means that agents tend to express high willingness to host the current VNF when all flags are up. At the same time, the counter-intuitive $p(2|0, 0, 0)$ (high willingness when all flags are down) and $p(0|1, 1, 1)$ (low willingness when all flags are up) seem to decrease over time. Our argument that these rules are indeed learned is backed by the fact that the respective probabilities diverge from the horizontal dotted line at $p = 0.33$, which indicates random action selection. Nevertheless, these lines never reach either 1.0 or 0.0, which may imply that the additional observation dimensions (besides the three flags) also play an important role in the action selection.

D. CROSS-AGENT COMMUNICATION

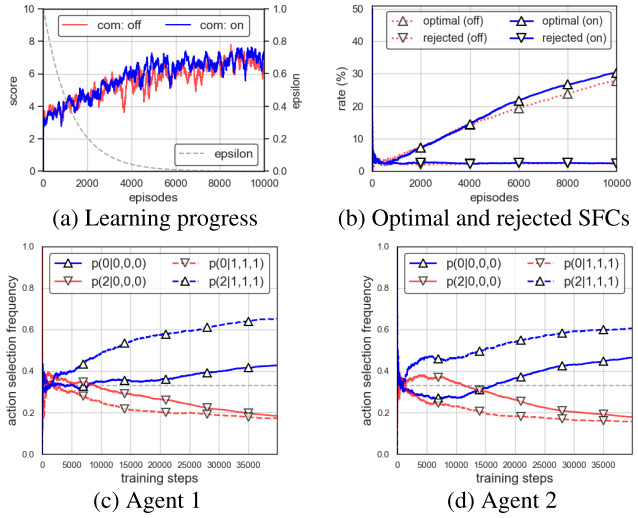
We now evaluate the impact of cross-agent communication on the overall efficiency of the DMARL scheme across all three PoP topologies shown in Fig. 4. Therefore, we implement a variant of the DMARL algorithm, where agents do not receive the neighbors' condensed state (*i.e.,* location and average available CPU in o_t^α).

Fig. 7 summarizes our findings for the mesh topology (Fig. 4a). According to Figs. 7a and 7b, the DMARL scheme with the communication feature *on* dominates the respective scheme with communication *off*. In particular, the former manages to converge faster towards an average score of 7.0, and its rate of optimal partitionings grows slightly quicker, compared to the latter. To interpret this performance difference, we lay out Figs. 7c and 7d for DMARL with communication *off*, meant to be compared with Figs. 6b and 6c respectively, where DMARL employs cross-agent communication. We focus on *Agent 1* and *Agent 2*, since they operate over two pivotal PoPs in the mesh topology (*i.e.,* PoPs 1 and 2 are highly relevant for SFCs). From the comparison of the respective $p(x|f_1, f_2, f_3)$-lines, we observe that, when



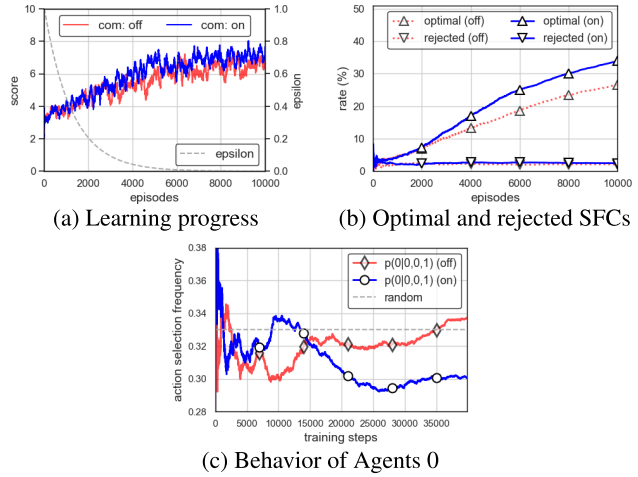
(a) Agent 0 (b) Agent 1 (c) Agent 2 (d) Agent 3 (e) Agent 4

FIGURE 6. Action selection rules identified by DMARL agents.



(a) Learning progress (b) Optimal and rejected SFCs
(c) Agent 1 (d) Agent 2

FIGURE 7. Cross-agent communication in the mesh topology.



(a) Learning progress (b) Optimal and rejected SFCs
(c) Behavior of Agents 0

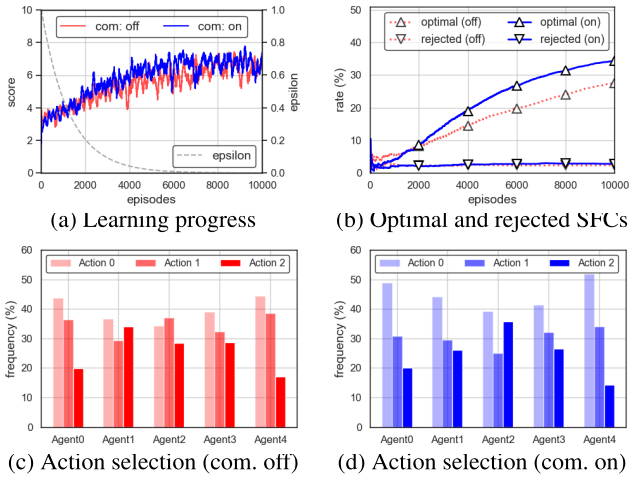
FIGURE 8. Cross-agent communication in the star topology.

cross-agent communication is disabled, action selection *rules* are learned both slower and less firmly.

Results from the respective comparison in the star topology (Fig. 4b) are depicted in Fig. 8. In particular, Figs. 8a and 8b emphasize the superiority of the DMARL algorithm that uses cross-agent communication. Here, the former converges both faster and to a higher average reward, and the difference in optimal partitionings is close to 10%. It is worth noting that, we observed slower and less firm identification of action selection *rules* in this case also, but the respective results are omitted due to space limitations. Instead, we plot Fig. 8c, where we zoom into a behavioral aspect of *Agent 0* (since PoP 0 is a pivotal PoP in the star topology). Specifically, $p(0|0,0,1)$ is lower for *Agent 0* when cross-agent communication is enabled, meaning that this agent is still more likely to select actions 1 or 2 when it is in the shortest path from the *src* to the *dst*, even if it does not host another VNF or the previous VNF. Note that the respective line for DMARL with communication *off* is closer to random. This corroborates the fact that enabled communication augments *Agent 0* to perceive its position in the star topology and, hence, to potentially facilitate SFCPs as an intermediate PoP.

Finally, we assess the cross-agent communication feature in the linear PoP topology shown in Fig. 4c. Here, results regarding the learning progress (Fig. 9a) and the rate of optimal and rejected SFCPs (Fig. 9b) follow a similar trend to that of the star topology, and the benefits of cross-agent communication remain. Some insightful micro-benchmarks regarding the difference of the two schemes are illustrated in Figs. 9c and 9d. Specifically, these bar charts indicate action selection frequencies per agent during the last 2, 000 episodes (where agents are essentially purely exploitative, as $\epsilon \rightarrow 0^{+}$). Based on Fig. 9c, *Agent 1* shows the most willingness to host VNFs; however, this is contradicting to the linear topology in which we would expect *Agent 2* to participate in more SFCPs. Therefore, *Agent 2* cannot perceive its pivotal position in the linear topology when the communication is *off*. In contrast, in Fig. 9d, notice that *action 2* frequencies are better aligned with the position of agents in the linear topology. Specifically, we observe that *Agent 2* is the most willing to allocate its PoP resources to VNFs, followed by its adjacent agents *1* and *3*, while agents *0* and *4* participate in less SFCPs, given their outermost position in the linear topology. We also note that, in principle, when communication is disabled, agents tend to select *action 1* (*i.e.,* neutral position with respect to hosting a VNF) more frequently, which can be interpreted as lack of confidence in their learning capacity.

According to the analysis above, cross-agent communication of short, yet informative, messages (i) enables faster and firmer identification of action selection rules, and



(a) Learning progress (b) Optimal and rejected SFCs

(c) Action selection (com. off) (d) Action selection (com. on)

FIGURE 9. Cross-agent communication in the linear topology.

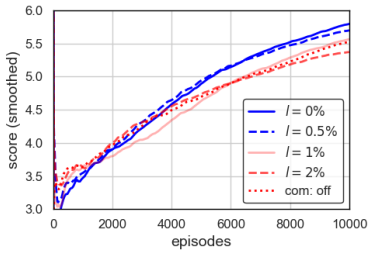


FIGURE 10. Imperfect cross-agent communication schemes.

(ii) enhances the agents' ability to identify their position within the topology and adjust their action selection behavior accordingly.

E. IMPERFECT CROSS-AGENT COMMUNICATION

The proposed DMARL scheme relies on the exchange of concise information among neighboring agents. Effectively, this is realized via cross-VIM/PoP communications (cf. **step 2** in Fig. 3). However, communications are hardly ever perfect. For example, data communication may be delayed, packets may be lost, or erroneous data may appear owing to lossy (de)compression or channel fluctuation. Moreover, the communication medium may not be dedicated for a single service; therefore, the opportunities for information exchange between neighboring agents will be limited. Yet, if cross-PoP communications are totally infeasible, one can resort to the DMARL variant with the cross-agent communication feature *off*, whose behavior has been analyzed in Section V-D. Despite of some performance degradation, it still manages to optimize SFCPs to a certain degree (cf. Figs. 7, 8, and 9).

To quantify the importance level of cross-agent communications, we evaluate our DMARL framework over lossy inter-PoP links[4] that exhibit non-zero packet loss rates l, (*i.e.,* $l > 0$). That is, an agent α receives nothing from its neighbor agent $\beta \in N(\alpha)$ with probability l (in this case, the respective entries in o_t^α are filled with 0s), while it receives the actual condensed state of agent β with probability $1 - l$. Here, we experiment with three levels of loss rate, *i.e.,* $l = 0.5\%$, $l = 1\%$, and $l = 2\%$, over the mesh topology (Fig. 4a).

In Fig. 10, to better identify their performance differences, we plot the respective scores as moving averages across the entire training period. The observations that can be drawn are as follows. First, smaller loss rates come with better DMARL performance, while even $l = 2\%$ loss rate leads to half unit lower average score compared to the perfect communication scheme (*i.e.,* where $l = 0\%$). Another interesting outcome concerns the comparison of the imperfect communication schemes with the communication *off* scheme. As per Fig. 10, disabling cross-agent communication (*i.e., com: off*) results in similar scores with the DMARL which operates over a mesh topology with $l = 1\%$ loss rate, and even outperforms the scenario with $l = 2\%$. That is because in the communication *off* scheme, agents learn optimized policies without ever expecting information from their neighbors. Conversely, in the communication *on* schemes, agent policies rely on cross-agent communications, and if the latter are faulty, this has an adverse effect on action selection. Further, the observation dimensions in communication *off* schemes are always less than the corresponding dimensions of communication *on* schemes. In this sense, communication *off* is not equivalent to communication *on* with $l = 100\%$.

At first glance, the above results suggest that the proposed DMARL scheme is sensitive to the loss of cross-agent communications and would require ultra-reliable communications of neighbors' states. However, this is not fully accurate. In fact, a variant of the above experiment - where each agent knows perfectly the location of its neighbors (*i.e.,* $v.loc, \forall v \in N(u)$ for PoP u), and the loss rate is only applied to the neighbors' average available CPU (*i.e.,* $d_{S,t}(v), \forall v \in N(u)$ for PoP u) - is conducted and only marginal differences are observed in terms of score compared to the case with ideal cross-agent communication. In conclusion, it is crucial for DMARL agents to be aware of the positions of their neighbors, as this position awareness allows them to perceive their own position within the topology more accurately and, in extension, to develop appropriate action selection behaviors. This aspect, *i.e.,* the significance of the content of the exchanged information, opens up the opportunity to apply different quality of service (QoS) requirements to different information of neighbors' states.

VI. RELATED WORK

This section includes a comprehensive discussion on recent works that employ reinforcement learning techniques for SFCP. In particular, we classify them into i) single-domain SFCP, where all PoPs belong to the same provider, and ii) multi-domain SFCP, where PoPs belong to multiple providers.

A. RL IN SINGLE-DOMAIN SFCP

Most studies that tackle the SFCP problem in single-domain settings opt for centralized solutions. In our context, these

[4]We here focus on the loss of neighbors' state information, rather than the packet loss of VNF data traffic.

are centralized RL agents handling decision-making over the entire set of PoPs across the domain.

In particular, Quang et al. [6] devise a deep deterministic policy gradient method based on the actor-critic RL paradigm. In order to enhance the exploration capacity of their method, authors adopt a *multiple critic networks* approach. These networks are intended to evaluate multiple noisy actions produced on the basis of the *promo* action selected by the actor network. At each time step, the true action to be selected is the one with the largest mean Q-value, computed across all critic networks. However, the actual updates on the actor network are driven by the critic network that exhibited the lowest loss. It is worth noting that the above RL scheme merely computes placement priorities, and not actual VNF allocations. The latter are handled by a heuristic termed as *heuristic fitting algorithm*, which maps VNFs onto physical nodes in a greedy fashion based on rankings and, subsequently, utilizes Dijkstra's method to compute link assignments.

Pei et al. [7] propose an SFCP framework which leverages DDQL. Here, the decision-making process is subdivided into three steps. First, given a network state, a preliminary evaluation of each action is performed, *i.e.,* computation of respective Q-values. Out of all actions, all but the k best are discarded. In the second step, these k actions are executed in simulation-mode, and the actual reward and next states are observed. Finally, the action with the highest reward is performed on the physical infrastructure.

Zheng et al. [8] investigate the deployment of SFCs in the context of cellular core. To this end, authors put both the intra-datacenter and the intra-server placement problem into the frame. Focusing our discussion on the intra-datacenter placement, authors devise an approximation algorithm for the optimized SFCP, under the assumption that the resource requirements and the lifespan of each SFC are fully disclosed. However, given the strictness of this hypothesis, authors implement an SFCP scheme that can cope with demand uncertainty, as well. In particular, the proposed method is conventional Q-learning. Here, a state s_t is expressed as the SFC type that needs to be deployed at time t, while an action x_t represents the selection of a server. Nonetheless, conventional Q-learning is not a viable option when the state-action space is vast.

Wang et al. [11] and Jia et al. [12] investigate SFCP through the lens of fault-tolerance, essentially considering the deployment of redundant VNFs or entire SFCs which shall be engaged in the event of malfunction in the running SFC. In particular, Wang et al. [11] devise a DDQL algorithm that is trained to compute an ordered pair (u, v) where u is the index of the DC for proper deployment, and v the backup DC where the standby SFC instance will be deployed. Effectively, this implies that the entire SFCs will be placed in a single PoP. Jia et al. [12] decompose the SFC scheduling problem with reliability constraints into two subproblems. First, they use a heuristic to determine the number of redundant VNF instances (per VNF). Then, they employ an A3C algorithm which, in conjunction with a rule-based node selection approach, facilitates the process of mapping these instances onto compute nodes. In practice, the DRL algorithm learns whether to defer or not the deployment of a VNF, instead of learning where to map it. However, given such a topology-independent action space, the proposed DRL scheme is not sensitive to evolving topologies.

In contrast to the above, our work promotes single-domain SFCP in a decentralized, yet coordinated fashion, addressing various limitations of existing studies. First, the proposed DMARL scheme empowers local controllers (*i.e.,* VIMs) with the ability to express their own resource utilization intents based on detailed local observations (as opposed to *e.g.,* [4], [5], [6], [7], [8], [9], [10], [11], [12]). This is explicitly achieved by designating one DDQL agent per VIM, and by utilizing action sets that express the degree of willingness with respect to hosting VNFs. Second, our DMARL scheme can indeed claim decision-making with complete information, since, from the perspective of the team of agents, the environment is fully observable. This is not the case with many studies, which either define incomplete representations of the true state or assume very detailed knowledge at the orchestration layer (*e.g.,* [4], [6], [7], [8], [10], [11]). Third, the action spaces used by individual DDQL agents of the DMARL framework are topology-agnostic (as opposed to *e.g.,* [4], [5], [6], [7], [8], [9], [10], [11]), which alleviates scalability limitations and renders our framework a promising candidate for dynamic topologies.

B. RL IN MULTI-DOMAIN SFCP

Multi-domain SFCP is commonly solved via distributed methods [17]. This can be attributed to the limited information sharing among different providers, which hinders the possibility of solving SFCP in a centralized fashion.

In [15], authors assign a DDPG-based RL agent to each domain. Further, the SFC request is encoded by the client (*i.e.,* the entity that wants to deploy the SFC) and conveyed to each RL agent. The latter then treat the SFC encoding as state and compute an action, which is effectively the bidding price for renting out their resources to the SFC request. Prices are accumulated by the client, who opts for the best combination based on a cost-based first fit heuristic. The MARL setting here is inherently competing, as agents do not have any incentive to maximize the rewards of other agents.

Toumi et al. [16] propose a hierarchical MARL scheme which employs a centralized agent (*i.e.,* multi-domain orchestrator - MDO) on top of multiple local-domain agents, where all agents are implemented with the DDQL architecture. The MDO receives the SFC request, and decides which local domain will host each constituent VNF. Afterwards, the agents of local domains are responsible for placing the sub-SFCs within their own nodes. Admittedly, the proposed MDO is quite similar to our own DDQL implementation for SFCP. If we assume that the partial observation of our DDQL due to data aggregation is equivalent to the partial observation of the MDO due to limited information sharing

across multiple domains, then we can think of the comparison in Section V-B as a comparison of DMARL with MDO.[5]

In [27], Zhu et al. devise a MARL scheme for SFCP over IoT networks. Specifically, they propose a hybrid architecture that employs both centralized training and distributed execution strategies. Here, the SFCP problem is modeled as a multi-user competition game model (*i.e.,* Markov game) to account for users' competitive behavior. In their implementation, the centralized controller performs global information statistical learning, while each user deploys service chains in a distributed manner, guided by a critic network.

While multi-domain SFCP is vertical to single-domain SFCP on the grounds of information disclosure, the related solutions exhibit some similarities to the proposed DMARL scheme. In effect, agents voting on hosting VNFs can be paralleled to agents bidding for resources, while our coordination module can be seen as a centralized broker which aggregates votes and computes the best action. However, the intrinsic competition that underlies multi-domain scenarios cannot be overlooked. For example, notice how agents in multi-domain SFCP strive to maximize their own rewards, disregarding the rewards of others. As such, our DMARL algorithm cannot be directly compared to any solution pertaining to the above.

VII. DISCUSSION

Our work exhibits certain shortcomings, which we deem important to discuss and clarify. With respect to the underlying algorithmic framework we opted for, which is IQL, we have already acknowledged its lack of convergence guarantees. The natural next step is to examine a DMARL algorithm for SFCP based on *centralized training - decentralized execution* schemes, such as the ones proposed in [24] and [25], where finding the optimal joint policy is theoretically guaranteed (under certain assumptions). Additional limitations which we identify mostly pertain to the analysis of our method, rather on the method itself. In particular, we evaluate DMARL over relatively small multi-PoP topologies. Yet, the intention of this work is to explore in-depth the learning behavior that is developed by a team of independent RL agents in order to cooperatively solve SFCP. To this end, keeping the topologies small, not only allows us to delve easier into the behavior of individual agents, but to demonstrate the respective findings as well (*e.g.,* Fig. 6).

VIII. CONCLUSION

Irrespective of the above constraints, both IQL and our evaluation methodology enable us to obtain valuable knowledge. Specifically, the comparison in Section V-B indicates that DMARL outperforms a centralized state-of-the-art SFCP method based on DDQL, which we mainly attribute to the full observability of the former against the partial observability of the latter. Concretely, DDQL fails to consolidate VNFs to the extend DMARL does, given the fact that it is not aware of the exact server capacities. We further corroborate that the team of agents manages to recognize certain intuitive action selection rules in Section V-C. This is highly crucial, especially considering efforts towards machine-learning explainability. Moreover, we identify that disabling the exchange of concise messages among neighboring agents limits their ability to perceive their position within the multi-PoP system, which has a negative effect on the developed action selection behaviors (Section V-D). Last, we quantify the performance drop of DMARL over multi-PoP systems which exhibit non-zero packet loss rates (Section V-E). The key takeaway here is that it is highly important for DMARL agents to at least be aware of the position of their neighbors, which implies that location coordinates play an important role in the individual action selection policies.

While this work sheds light on numerous fundamental aspects of distributed resource allocation within NFV, at the same time it paves the way for several new research directions. For instance, having established the perks of decentralized SFCP, a natural next step is to examine a multi-agent RL algorithm at which agents shall learn *what*, *when*, *to whom* and *how* to communicate; in other words, agents shall learn their own cross-agent communication protocol.

[5] We acknowledge that these two sources of partial observability are not equivalent, hence a direct comparison of DMARL with MDO makes little sense.

REFERENCES

[1] M. Rost and S. Schmid, "On the hardness and inapproximability of virtual network embeddings," *IEEE/ACM Trans. Netw.*, vol. 28, no. 2, pp. 791–803, Apr. 2020.

[2] D. Dietrich, A. Abujoda, A. Rizk, and P. Papadimitriou, "Multi-provider service chain embedding with nestor," *IEEE Trans. Netw. Service Manage.*, vol. 14, no. 1, pp. 91–105, Mar. 2017.

[3] A. Pentelas, G. Papathanail, I. Fotoglou, and P. Papadimitriou, "Network service embedding across multiple resource dimensions," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 1, pp. 209–223, Mar. 2021.

[4] S. Haeri and L. Trajković, "Virtual network embedding via Monte Carlo tree search," *IEEE Trans. Cybern.*, vol. 48, no. 2, pp. 510–521, Feb. 2018.

[5] 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. Int. Symp. Quality Service*, Jun. 2019, pp. 1–10.

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

[7] 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.

[8] J. Zheng, C. Tian, H. Dai, Q. Ma, W. Zhang, G. Chen, and G. Zhang, "Optimizing NFV chain deployment in software-defined cellular core," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 2, pp. 248–262, Feb. 2020.

[9] R. Solozabal, J. Ceberio, A. Sanchoyerto, L. Zabala, B. Blanco, and F. Liberal, "Virtual network function placement optimization with deep reinforcement learning," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 2, pp. 292–303, Dec. 2020.

[10] 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.

[11] L. Wang, W. Mao, J. Zhao, and Y. Xu, "DDQP: A double deep Q-learning approach to online fault-tolerant SFC placement," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 1, pp. 118–132, Mar. 2021.

[12] J. Jia, L. Yang, and J. Cao, "Reliability-aware dynamic service chain scheduling in 5G networks based on reinforcement learning," in *Proc. IEEE Conf. Comput. Commun.*, May 2021, pp. 1–10.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and G. Ostrovski, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.
[14] D. Dietrich, A. Rizk, and P. Papadimitriou, "Multi-domain virtual network embedding with limited information disclosure," in *Proc. IFIP Netw. Conf.*, 2013, pp. 1–9.
[15] 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 Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2019, pp. 886–891.
[16] N. Toumi, M. Bagaa, and A. Ksentini, "Hierarchical multi-agent deep reinforcement learning for SFC placement on multiple domains," in *Proc. IEEE 46th Conf. Local Comput. Netw. (LCN)*, Oct. 2021, pp. 299–304.
[17] J. C. Cisneros, S. Yangui, S. E. P. Hernández, and K. Drira, "A survey on distributed NFV multi-domain orchestration from an algorithmic functional perspective," *IEEE Commun. Mag.*, vol. 60, no. 8, pp. 60–65, Aug. 2022.
[18] F. Carpio, S. Dhahri, and A. Jukan, "VNF placement with replication for loac balancing in NFV networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2017, pp. 1–6.
[19] A. Pentelas and P. Papadimitriou, "Network service embedding for cross-service communication," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, May 2021, pp. 424–430.
[20] A. Pentelas and P. Papadimitriou, "Service function chain graph transformation for enhanced resource efficiency in NFV," in *Proc. IFIP Netw. Conf. (IFIP Networking)*, Jun. 2021, pp. 1–9.
[21] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.
[22] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proc. 10th Int. Conf. Mach. Learn.*, 1993, pp. 330–337.
[23] L. Matignon, G. J. Laurent, and N. Le Fort-Piat, "Independent reinforcement learners in cooperative Markov games: A survey regarding coordination problems," *Knowl. Eng. Rev.*, vol. 27, no. 1, pp. 1–31, Feb. 2012.
[24] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel, "Value-decomposition networks for cooperative multi-agent learning," 2017, *arXiv:1706.05296*.
[25] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson, "QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4295–4304.
[26] J. Foerster, N. Nardelli, G. Farquhar, T. Afouras, P. H. Torr, P. Kohli, and S. Whiteson, "Stabilising experience replay for deep multi-agent reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1146–1155.
[27] Y. Zhu, H. Yao, T. Mai, W. He, N. Zhang, and M. Guizani, "Multiagent reinforcement-learning-aided service function chain deployment for Internet of Things," *IEEE Internet Things J.*, vol. 9, no. 17, pp. 15674–15684, Sep. 2022.

ANGELOS PENTELAS (Student Member, IEEE) received the B.Sc. degree in mathematics from the Aristotle University of Thessaloniki, Greece, and the M.Sc. degree in applied informatics from the University of Macedonia, Greece, where he is currently pursuing the Ph.D. degree. From October 2021 to March 2022, he was with Nokia Bell Laboratories, Belgium, as a Ph.D. Intern. His research interest includes decision-making methods for network orchestration.

DANNY DE VLEESCHAUWER (Member, IEEE) received the M.Sc. degree in electrical engineering and the Ph.D. degree in applied sciences from Ghent University, Belgium, in 1985 and 1993, respectively. He is currently a DMTS with the Network Systems and Security Research Laboratory, Network Automation Department, Nokia Bell Laboratories, Antwerp, Belgium. Prior to joining Nokia, he was a Researcher with Ghent University. His early work was on image processing and on the application of queuing theory in packet-based networks. His current research interest includes the distributed control of applications over packet-based networks.

CHIA-YU CHANG (Member, IEEE) is currently a Senior Research Engineer with Nokia Bell Laboratories, Belgium. As the Principal Investigator and a Technical/Research Lead, he has participated in a number of national and international research and innovation projects, focusing on B5G/6G communication systems, network intelligence solutions, network applications proof-of-concept among various vertical industries, and recently on low-latency low-loss scalable throughput (L4S) over a variety of access network, utility-based congestion control, and 6G network-application interaction. He was a Best Paper Award Winner of SDS'21 and MediaTek Innovation Award Winner, in 2013.

KOEN DE SCHEPPER (Associate Member, IEEE) received the M.Sc. degree in industrial sciences (electronics and software engineering) from IHAM Antwerpen, Belgium. He joined Nokia (then Alcatel), in 1990, where during the first 18 years, he was the Platform Development Leader and a Systems Architect. He has been with Bell Laboratories for the past 14 years. He is currently with the Network Systems and Security Research Laboratory. Before, he worked mainly on transport layer protocols (L4) and their network support for scalable (SCAP) and low latency (L4S) content delivery. His current research interests include programmable data plane and traffic management, customizable network slicing, and AI supported dynamic service and network control.

PANAGIOTIS PAPADIMITRIOU (Senior Member, IEEE) received the B.Sc. degree in computer science from the University of Crete, Greece, in 2000, the M.Sc. degree in information technology from the University of Nottingham, U.K., in 2001, and the Ph.D. degree in electrical and computer engineering from the Democritus University of Thrace, Greece, in 2008. He is currently an Associate Professor with the Department of Applied Informatics, University of Macedonia, Greece. Before that, he was an Assistant Professor with the Communications Technology Institute, Leibniz Universität Hannover, Germany, and a member of the L3S Research Center, Hanover. His research interests include (next-generation) internet architectures, network processing, programmable dataplanes, time-sensitive networking (TSN), and edge computing. He has been a (co-)PI in several EU-funded (e.g., NEPHELE, T-NOVA, CONFINE, and NECOS) and nationally-funded projects (e.g., G-Lab VirtuRAMA and MESON). He was a recipient of the Best Paper Awards at IFIP WWIC 2012 and IFIP WWIC 2016, and the runner-up Poster Award at ACM SIGCOMM 2009. He has co-chaired several international conferences and workshops, such as IFIP/IEEE CNSM 2022, IFIP/IEEE Networking TENSOR 2021–2020, IEEE NetSoft S4SI 2020, IEEE CNSM SR+SFC 2018–2019, IFIP WWIC 2017–2016, and INFOCOM SWFAN 2016. He is also an Associate Editor of IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT.