Deep Reinforcement Learning Approaches to Network Slice Scaling and Placement: A Survey

Niloy Saha, Mohammad Zangooei, Morteza Golkarifard, and Raouf Boutaba

The authors survey various DRL-based approaches to slice scaling and placement, including different ways to model the problem and benefits of various DRL techniques in addressing specific aspects of the problem.

Abstract

Network slicing in 5G and beyond networks allows the network to be customized for each application or service by chaining together different virtualized network functions (VNFs) according to service requirements. The increased flexibility offered by network slicing comes at the cost of complexity in management and orchestration, which cannot be solved by traditional reactive human-in-the-loop solutions. This necessitates minimizing human intervention through the use of artificial intelligence techniques (zero touch network management). In particular, the scaling and placement of the chain of VNFs that constitute a network slice is a complex combinatorial optimization problem that is difficult to solve effectively with traditional approaches. Driven by the benefits of deep reinforcement learning (DRL) in solving various combinatorial optimization problems, in this article, we survey various DRL-based approaches to slice scaling and placement, including different ways to model the problem and benefits of various DRL techniques in addressing specific aspects of the problem. Further, we highlight key challenges and open issues in the effective use of DRL for network slice scaling and placement.

Introduction

5G and beyond mobile networks are expected to support a variety of emerging use cases, such as holographic telepresence and immersive extended reality, which impose strict requirements on the mobile network along several dimensions, such as throughput, latency, and reliability. To address these requirements, network slicing in 5G allows the network to be customized for each application or service by chaining together different virtualized network functions (VNFs). The creation of network slices involves the instantiation and deployment of a chain (or network) of VNFs at the right location (placement) and with the appropriate capacity (scaling) to meet stringent requirements imposed by 5G services. The resources allocated to network slices must conform to the dynamics of network demands, which are unpredictable and vary with time and user mobility. Thus, the complexity in the management and orchestration of network slices makes it necessary to reduce human intervention and rely on automated zero -touch network management approaches.

Network Slice Scaling and Placement

A network slice comprises a chain VNFs with associated service requirements (e.g., end-to-end delay or throughput). The network slice scaling and placement (NSSP) problem involves instantiation or reconfiguration of a network slice by deciding the resource allocation and placement of each VNF in the slice. The scaling step requires deciding how many instances of a particular VNF to create and how much resources to allocate to each instance, while the placement step requires deciding on which physical node to place each instance of a VNF.

Figure 1 shows a generic network slice diagram, consisting of two slices. Consider Slice 1, comprising three network functions (NFs), NF1 $\rightarrow$ NF2 $\rightarrow$ NF3. These NFs depend on requirements of the slice: NFs can be security-related (e.g., firewall), virtualized RAN functions, or 5G core functions. Each slice also includes associated service requirements; for example, ultra-reliable low-latency communications (URLLC) services in 5G impose stringent latency (<1 ms) and reliability (> 99.999 percent) requirements. The NSSP problem involves taking as input this chain of NFs and deciding the embedding at the infrastructure layer, while respecting the slice constraints in terms of service level agreements (SLAs).

Traditional Approaches for NSSP

Most existing approaches formulate NSSP as an optimization problem [1], which has several limitations. First, these approaches are often intractable or too computationally intensive to be applied in practical settings. Second, they rely on accurate mathematical models that assume complete knowledge of the state of the environment, which is very difficult to obtain in practice. Finally, these methods also assume that traffic demands are predictable or known a priori, which is unrealistic. To address these challenges, several heuristic approaches also exist in the literature; however, they disregard the long-term dynamics of resource requests and can lead to frequent reconfigurations [2].

Deep Reinforcement Learning for NSSP

Closed-loop automation is crucial in realizing zero touch orchestration and management of network slices. Deep reinforcement learning (DRL) is an intuitive fit for closed-loop automation; it is an iterative process that uses feedback from the environment to

Digital Object Identifier: 10.1109/MCOM.006.2200534

The authors are with The University of Waterloo, Canada.

learn the correct sequence of actions to maximize a long-term reward (e.g., operator revenue, energy and resource costs). Unlike other artifical intelligence (AI) techniques (e.g., supervised learning) which are myopic in nature, DRL can intelligently adapt decisions to variations in the requirements over time. When applied to NSSP, DRL enables closed-loop optimization and control. At each time step, the scaling and placement decisions of VNFs output by DRL are used by the network orchestrator to modify the VNF embedding at the infrastructure layer without the need for human intervention. The reward obtained from this process is then used in the next step for further improving the decisions, thus facilitating closed-loop optimization.

From an optimization perspective, NSSP involves taking an optimal action from a large finite set (often discrete) of actions, and is, in essence, a combinatorial optimization problem. In recent works, DRL has been shown to be very effective in addressing such optimization problems [3] by automating the discovery of good heuristics. Crafting good heuristics is time-consuming and often requires substantial problem-specific knowledge. DRL algorithms automate the discovery of good heuristics by tailoring the search strategies to the problem instance using a data-driven approach. DRL approaches can be either *model-based* or *model-free*. Model-based approaches require an accurate model of the environment for effective learning. However, creating an accurate model of the end-to-end (E2E) network slicing environment is not cost-effective since a) the model may have a huge number of configurable parameters across multiple technological domains, and b) the model is dependent on time-varying incoming slice demands. On the other hand, model-free reinforcement learning techniques have the capability to learn with continuous interactions with the environment, without a priori knowledge of the network model or network statistics. Thus, the existing approaches in the literature [4–11] adopt model-free DRL approaches for NSSP.

A few existing surveys in the literature, such as [12], provide valuable insights regarding the efficacy of DRL in the broader topic of resource allocation for 5G network slicing, including radio resource slicing and slice admission control. In contrast, in this survey, we present a more focused look at NSSP by analyzing and comparing the state of the art using criteria specific to this problem such as VNF chaining and topology awareness.

DRL for NSSP: Problem Formulation

DRL enables data-driven learning by interacting directly with the environment and getting feedback in the form of rewards. Over time, the DRL agent can learn the underlying dynamics of a system, and leverage that to discover optimal strategies. To apply DRL to NSSP, it is first modeled as a sequential decision making process. More specifically, as shown in Fig. 2, the agent (5G management and orchestration framework) and the environment (5G network) interact at discrete time steps. At each step, the agent obtains some information about the environment *state* (e.g., traffic demand of each slice, CPU/memory utilization of each VNF in a slice), based on which the agent takes an *action* (e.g., add/remove resources to a VNF in a slice, or add/remove more instances of a VNF, or where to place a particular VNF). This leads to the agent obtaining some form of feedback (e.g., reward such as revenue from slice users) from the environment, giving rise to a trajectory or sequence consisting of (*state*, *action*, *reward*, *next state*). By repeating this process of interaction, the DRL agent learns a *policy*, that is, a strategy that dictates actions as a function of the state, in order to achieve long-term goals.

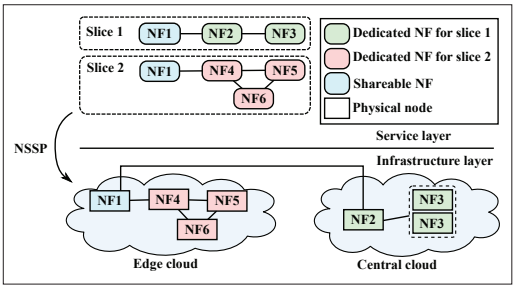

FIGURE 1. A generic network slice diagram showing two slices. Each slice is composed of VNFs some of which may be dedicated to the slice or shared among slices. Each VNF may have multiple instances and can be placed at different candidate locations in the network including central cloud or distributed edge cloud.

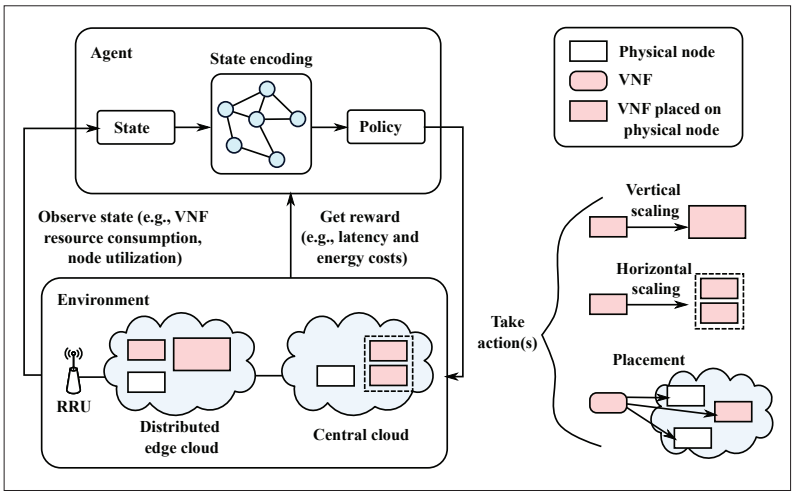

FIGURE 2. A high-level architecture showing DRL applied to the network slice scaling and placement problem. The figure shows the DRL agent's interaction with the environment (5G network). The state encoder in DRL algorithms are generally deep neural networks (DNNs) which act as non-linear function approximators for the network state. Actions represent resource scaling and placement decisions for VNFs in the network slice.

Modeling Approaches

The sequential decision making process described above is conveniently modeled using the Markov decision process (MDP) and its variants.

- *Markov decision process.* In the context of RL, an MDP may be formally defined by the tuple consisting of a set of states, a set of actions, and a set of rewards, and the goal is to find the policy that maximizes the expected sum of the rewards.
- *Semi Markov decision process.* Slice requests do not arrive at fixed intervals of time, but rather at random points in time, which may be taken into account by considering semi Markov decision processes (SMDPs). SMDPs extend the MDP formalism by incorporating the notion of time, which allows the agent to handle trade-offs between actions not only based on expected rewards but also on the amount of time each action takes. This allows

Ref.	DRL technique	State	Action	Reward
[4]	Monte Carlo	Resource consumption of VNF service chain	Increase or decrease VNF resources by discrete step sizes and offload VNF	Two-step reward consisting of throughput-to-latency ratio of a slice followed by SLA satisfaction ratio
[5]	DQN	VNF resource consumption, host node utilization	Add, remove, or maintain number of instances for VNF chain	Weighted sum of response time and resource consumption of VNF chain
[6]	DDQN with BDQ	Available resources of candidate paths, slice traffic demands, slice reconfiguration cost	Select path (sequence of host nodes) to place entire VNF chain	Weighted sum of slice resource consumption and slice reconfiguration cost
[7]	A3C and auxiliary tasks	Packet arrival rate to each slice, VNF resource consumption, queue length at VNF	Allocate resources to VNFs at each physical node	Utility function of delay between two VNFs in a chain
[8]	A3C with GCN augmented with heuristics	Number of VNFs placed on host node, host node resource utilization	Index of physical node to place each VNF of a slice	Utility function of slice acceptance, resource consumption, and load balancing
[9]	Variant of TD3	VNF resource consumption of each slice, arrival rate at each VNF	Select VNF location followed by allocation of resources	Weighted sum of latency (VNF resizing, deployment, or offloading to cloud) and cost of VNF deployment
[10]	TD3	Number of users and VNFs in each slice, energy and latency of each slice	Scale resources allocated to each VNF in a slice	Weighted sum of latency, compute, and energy costs
[11]	PPO	Existing and free compute resources, arrival rate of user sessions	Scale number of user plane VNFs	Weighted sum of request blocking rate and number of VNFs

TABLE 1. Summary of RL Techniques for NSSP.

the SMDP framework to be more effective in capturing real-time slice scaling events [13].

- *Constrained Markov decision process.* Network slice scaling and placement requires the agent to consider constraints (e.g., E2E latency) that may restrict the freedom of exploring the search space. This notion can be captured using constrained Markov decision processes (CMDPs) [14]. CMDPs extend traditional MDP by allowing the environment to provide feedback about the cost of constraints in addition to a reward signal. This formulation is well suited to NSSP, where reward should be maximized while maintaining several slice constraints, such that they do not violate SLAs.

DRL Setup

To formulate the NSSP problem as an MDP, in general, the following elements are defined:

- *State representation.* The majority of the existing literature uses a similar state representation, consisting of resource-related features such as VNF resource consumption and host node utilization [4–6, 8, 10]. A few works also consider features impacting slice latency such as packet arrival rate and queue length [7, 9, 11].
- *Action representation.* The action representation may be discrete, continuous, or hybrid, depending on the type of action. Horizontal scaling, which involves increasing/decreasing the number of active VNF instances, or placement, which involves selecting a host node to place VNFs, gives rise to discrete action spaces [5, 6]. On the other hand, vertical scaling, which involves the addition/removal of resources from existing VNFs, gives rise to continuous action spaces [7, 10]. Addressing both scaling and placement jointly requires the use of both continuous and discrete (i.e., hybrid action spaces) [4, 9].
- *Reward function.* The majority of the existing literature use scaling and placement cost as the reward function. The costs are usually calculated in terms of resource or energy consumption for running VNFs belonging to a network slice [5, 8] or blocking rate of slice requests [11]. The reward function may also include penalties for violating SLA constraints [5, 8, 9].

We summarize the various state, action, and reward representations used in the existing literature in Table 1. Once NSSP has been expressed as an MDP (or its variants), various DRL techniques may be utilized to find an optimal policy. These are discussed next.

DRL for NSSP: Algorithms

The majority of the existing literature on NSSP leverages model-free DRL algorithms, which learn through continuous interactions with the environment without a priori knowledge of the network model or network statistics. At a high level, we classify them into *value-based* approaches and *actor-critic* approaches. The existing works in the literature all formulate the MDP in slightly different ways and use a variety of algorithms, which are summarized in Table 1.

Value-Based Approaches

Value-based approaches involve finding the optimal policy by approximating the value of taking an action in a given state, and choosing the best action based on this approximation. This approach lends itself well to discrete action spaces such as horizontal scaling or placement. One of the simplest value-based methods is Monte Carlo, used in [4], where the state-action value is the average of rewards for each episode. Here, an episode is the sequence of steps involved in increasing/decreasing the resources for each VNF. Another simple value-based approach is the deep Q network (DQN), which utilizes a neural network to approximate the state-value function. In the context of NSSP, Lee *et al.* [5] utilized DQN to address horizontal scaling. Here, the state-value function is approximated using a two-layer multi-layer perceptron (MLP), which acts as the state encoder. This approach is shown to have much better long-term performance than simple threshold-based scaling; however, the authors did not consider the placement

of the VNF instances. The combination of scaling and placement decisions in NSSP leads to a combinatorially large discrete state/action space, which is not well suited for the application of simple DQN.

This issue is tackled by [6]; rather than scaling coarse-grained network slices according to traffic demand, they consider very fine-grained network slices, with each traffic flow mapped to a network slice. The goal is to map the flows to specific slices according to their demands in order to minimize long-term resource consumption. In this setting, the action space can be very large as it is combinatorial in the number of flows and slices. To address this, the authors adopt the branching dueling Q-network (BDQ) framework. The core idea behind BDQ is to avoid a combinatorial increase in action space by having independent actions for each dimension. The action branching framework allows the actions to increase linearly with the number of dimensions of the action space, which is useful for NSSP.

Actor-Critic Approaches

In contrast to value-based approaches, policy-based approaches, which learn the policy directly, can support continuous action spaces. However, they suffer from low sample efficiency, high variance, and slow convergence. Actor-critic approaches combine the benefits of both value and policy approaches by simultaneously learning the value and policy function. A popular actor-critic algorithm is advantage actor-critic (A3C), used in [7], which focuses on resource allocation among VNF chains. Here, the action space is continuous — an action is defined as allocating a certain ratio of resources to a given flow. To increase the learning speed and robustness of A3C, the authors adopted the concept of *auxiliary tasks*. The intuition behind auxiliary tasks is to add additional learning goals to the agent that are used to optimize the feature extraction pipeline in a way that may be useful for the given task and can significantly increase the sample efficiency. The authors in [8] also leverage the A3C algorithm, but incorporate a few notable changes. First, they include a graph convolutional network (GCN) to extract topological features from the substrate network. Second, to accelerate the convergence of A3C, they introduce an efficient placement heuristic.

Another popular actor-critic algorithm is deep deterministic policy gradient (DDPG), which extends ideas of DQN to continuous action spaces. Twin delayed DDPG (TD3) is an evolution of DDPG and includes improvements to stabilize learning and reduce sensitivity to hyperparameter variation. TD3 is used in [9, 10]. The authors in [10] focus on vertical scaling (i.e., adding/removing computing resources allocated to each VNF according to dynamic traffic fluctuations), while the authors in [9] consider both horizontal and vertical scaling. The authors in [10] show that TD3 performs well in terms of respecting latency constraints, due to more stable learning.

Proximal policy optimization (PPO) is another approach that can be applied to both continuous as well as discrete action spaces, and is used in [11] to scale the number of user plane VNFs according to the arrival rate of user sessions. PPO includes a built-in mechanism to avoid changing the training parameters too much in a single step, leading to more stable training. The authors in [11] show that PPO learns a stable policy more consistently than DQN in trading off between the number of VNFs and blocking rate of requests.

Synthesis: Several works in the existing literature show the efficacy of using DRL approaches for NSSP. Table 1 provides a qualitative comparison of the state-of-the-art in terms of modeling NSSP as a DRL problem, and the DRL algorithms used. Recent value-based DRL algorithms [4–6] are a good approach to horizontal resource scaling with discrete action spaces. However, the NSSP problem includes both continuous and discrete action spaces, and may be better served by using action-critic approaches. Most actor-critic approaches in the literature [7, 8, 10, 11] focus on either scaling or placement, apart from [9], which considers both. Among the actor-critic approaches in the literature used to address NSSP, TD3 [10] is more sample-efficient compared to PPO [11]. Sample efficiency is crucial in 5G networks, as the reward signal — generally obtained by monitoring, is associated with significant overheads. The A3C method [8] is very fast by virtue of parallelism; however, it suffers from instabilities in learning. The introduction of auxiliary tasks [7] attempts to address this; however, designing good auxiliary tasks is problem-dependent. The majority of the existing works also lack a comparison of the proposed DRL method with traditional methods (e.g., optimization approaches), which makes it hard to judge the extent of benefit DRL provides. One reason for this is the network scale; optimization-based approaches struggle when the scale is large, making a comparison of practical large-scale networks difficult.

Challenges in Effective Use of DRL for NSSP

In this section, we discuss the challenges in effectively using DRL for NSSP. Table 2 summarizes the effectiveness of DRL approaches in the existing literature in addressing these challenges.

VNF Chaining

The next phase of development for 5G and beyond has seen a push toward cloud native technologies, where NFs are decomposed into simpler NF services (referred to as VNF components or VNFCs) and are implemented using lightweight virtualization mechanisms (e.g., Linux containers or Unikernels). A network slice may consist of a chain of these containerized VNFCs, each of which may have multiple instances and need to be placed in a particular order. Most of the literature considers individual VNFs in a chain to be placed and scaled independently [4, 6, 9–11], or do not consider the ordering of VNFs in the chain [5, 7, 8]. This has a range of implications:

- An overloaded VNFC can cause a cascading domino effect on the subsequent VNFCs in the chain, thus triggering redundant scaling operations
- VNFCs belonging to the same VNF may have frequent communication with each other, which must be taken into account while performing placement; otherwise, it may lead to significant communication overhead.

Thus, the DRL approaches need to be augmented with mechanisms to incorporate VNF chaining information along with the state representation.

The next phase of development for 5G and beyond has seen a push toward cloud native technologies, where NFs are decomposed into simpler NF services (referred to as VNF components or VNFCs) and are implemented using lightweight virtualization mechanisms (e.g., Linux containers or Unikernels).

Ref.	Placement	Scaling	VNF chaining	Topology awareness	Constraint awareness	Evaluation
[4]	Partial, placement between different edge sites	Yes	No	No	Reward shaping using slice throughput and latency	Emulated VNF testbed based on Docker
[5]	No	Yes	Partial, no ordering	No	Slice latency used as penalty	OpenStack NFV testbed
[6]	Yes	No	No	No	VNF capacity and link bandwidth constraints	Simulation
[7]	Fixed	Yes	Partial, no ordering	No	Reward shaping using delay and throughput	Trace-driven simulation
[8]	Yes	No	Partial, each VNF in a chain placed independently	Yes	Slice placement failure used as penalty	Python-based simulation
[9]	Yes	Yes	No	No	QoS violation per VNF used as penalty	Simulation
[10]	No	Yes	No	No	Reward shaping using weighted sum of computation, latency, and energy cost	Custom slicing environment using OpenAIGym
[11]	No	Yes	No	No	No	Python-based simulation of container environment

TABLE 2. Analysis of existing DRL literature in addressing challenges for NSSP.

Topology Awareness

Apart from the VNF chain structure, it is also important to consider the physical network topology. The VNFs in a network slice can be placed at different network function virtualization infrastructure (NFVI) points of presence, including the central cloud and various distributed edge clouds. The placement of NFs and the physical network topology may have several implications in terms of communication overhead. For example, VNFs that frequently communicate with each other can be placed at the same node. This reduces frequent information exchange over the network, thus reducing communication overhead. A DRL agent usually encodes the state information in the form of a vector. Most existing DRL approaches use some type of deep neural network (DNN) [5] as the encoder, while the topology information is in the form of a graph. As such, they do not make use of the network topology information. In this regard, graph neural networks (GNNs) can be leveraged to preserve the topological dependencies by encoding the graph relationships [8]. GNNs take the topology graph as an input and generate embedding vectors to capture the essential features of the topology.

Design Challenges

The majority of the existing literature on slice scaling and placement considers either vertical (continuous action space) or horizontal scaling (discrete action space) in isolation and applies the appropriate class of DRL, such as actor-critic or DQN. For more flexible and granular resource scaling, we need to jointly consider both horizontal and vertical scaling. This gives rise to hybrid (continuous and discrete) action space in DRL formulations, which cannot be addressed by the majority of popular DRL algorithms. Recent attempts to tackle hybrid action spaces include parameterized DQN, where the agent selects discrete actions, each associated with a continuous set of parameters [15].

The state encoders for DRL typically include DNNs with fixed input and output dimensions based on the states and actions. During training, the dimensions of the input and output layers usually stay unchanged. This implies that when the conditions change (e.g., network topology change or change in length of VNF chain), the input and output layers need to be changed to reflect the updated state and action space, and the new model needs to be re-trained from scratch. Several neural network approaches allow handling variable input — recurrent neural networks such as long short-term memory (LSTMs) or gated recurrent units (GRUs) may be used to work with mixed length sequences, while GNNs may be used to work with arbitrarily sized graphs. However, further investigations are required to determine their efficacy in the context of DRL in terms of data efficiency and generalization performance.

Constraint Awareness

DRL approaches usually consider random exploration mechanisms for performing policy improvement over time. However, random exploration mechanisms for scaling and placement decisions may degrade the performance of a slice enough to violate the SLAs. Thus, it is crucial to include constraint awareness in DRL, as a violation of SLAs can make DRL approaches impractical for real-world implementation. One approach to address this is to consider constrained RL using mechanisms such as reward shaping (i.e., using constraint violations as penalties in the reward function) [4, 7, 10]. In this context, proper design of the reward function is crucial — too little weightage to penalty terms can cause SLA violations, while too much weightage can significantly slow down exploration and policy improvement. Further, depending on the problem scenario, it may be important to design the reward function to consider the performance of the worst case user instead of the average reward.

Evaluation Challenges

DRL agents learn by directly interacting with the environment and need a large number of interactions to collect sufficient information to train DNN structures. This involves the collection of a large

number of network measurements from various segments of the network such as radio access, edge, and core. Depending on the number of parameters and measurement frequency, it may be impractical to send all measurements to the DRL agent while maintaining reasonable network overhead. The majority of the literature on NSSP leverage simulations that do not account for this network overhead [6–9]. Additionally, in practice, it may not be possible to collect certain state variables related to VNFs due to privacy or security reasons. Thus, it may be necessary to operate on only a subset of these measurements and leverage modeling approaches such as the partially observable MDP (POMDP), in which the underlying state cannot be directly measured.

Further, the scope of evaluation in most existing work is restricted to simulations; extensive evaluations are needed using a real-world testbed to judge the effectiveness of these DRL approaches in practice. In this regard, there are several open source projects (e.g., srsRAN) that can be used to implement a real 5G testbed; however, to make the application of DRL on top of these viable, challenges related to monitoring (collecting state information from the environment) and orchestration (translating DRL actions to actual scaling and placement decisions) need to be addressed. Another challenge is related to the high sample complexity of DRL. DRL agents learn by interacting with the environment, and in practice, the orchestration interval (time taken for VNF migration and network device reconfiguration) can be quite high. Thus, each iteration of the DRL algorithm can be time-consuming, leading to a long convergence time. One approach to address this is to leverage simulations for generating enough training samples and fine-tuning the trained model on a real-world testbed.

Conclusion

Network slice scaling and placement is a fundamentally challenging combinatorial optimization problem. Previous approaches in the literature have included integer programming or custom-designed heuristics; however, recent work has shown that DRL can prove to be a promising alternative approach. DRL can use neural networks to learn the underlying structure and leverage the learned problem structure to search for an optimal policy. This is in line with exploiting the problem structure to design custom heuristics. However, DRL automates the process of finding an optimal policy by adopting a data-driven approach and avoids the need to manually design and tune such heuristics. However, DRL is not a panacea — in this article, we have discussed several challenges that need to be addressed for the effective use of DRL for network slice scaling and placement.

References

[1] D. Harutyunyan, R. Behravesh, and N. Slamnik-Kriještorac, "Costefficient Placement and Scaling of 5G Core Network and MEC-Enabled Application VNFs," *Proc. IFIP/IEEE Int'l. Symp. Integrated Network Management*, May 2021, pp. 241–49.

[2] J. J. Alves Esteves *et al.*, "Heuristic for Edge-Enabled Network Slicing Optimization Using the 'Power of Two Choices'," *2020 16th Int'l. Conf. Network and Service Management*, Nov. 2020, pp. 1–9.

[3] N. Mazyavkina *et al.*, "Reinforcement Learning for Combinatorial Optimization: A Survey," *Computers & Operations Research*, vol. 134, 2021, p. 105,400; https://www.sciencedirect.com/science/article/pii/S0305054821001660.

[4] M. Nakanoya, Y. Sato, and H. Shimonishi, "Environment-Adaptive Sizing and Placement of NFV Service Chains with Accelerated Reinforcement Learning," *Proc. IFIP/IEEE Symp.n Integrated Network and Service Management*, 2019, pp. 36–44.

[5] D. Lee, J.-H. Yoo, and J. W.-K. Hong, "Deep Q-Networks Based Auto-Scaling for Service Function Chaining," *Proc. Int'l. Conf. Network and Service Management*, 2020, pp. 1–9.

[6] F. Wei *et al.*, "Network Slice Reconfiguration by Exploiting Deep Reinforcement Learning with Large Action Space," *IEEE Trans. Network and Service Management*, vol. 17, no. 4, 2020, pp. 2197–2211.

[7] N. Yuan *et al.*, "Delay-Aware NFV Resource Allocation with Deep Reinforcement Learning," *Proc. IEEE/IFIP Network Operations and Management Symp.*, 2020, pp. 1–7.

[8] J. J. A. Esteves *et al.*, "A Heuristically Assisted Deep Reinforcement Learning Approach for Network Slice Placement," *IEEE Trans. Network and Service Management*, 2021.

[9] J. S. Pujol Roig, D. M. Gutierrez-Estevez, and D. Gündüz, "Management and Orchestration of Virtual Network Functions via Deep Reinforcement Learning," *IEEE JSAC*, vol. 38, no. 2, 2020, pp. 304–17.

[10] F. Rezazadeh *et al.*, "Continuous Multi-Objective Zero-Touch Network Slicing via Twin Delayed DDPG and OpenAI Gym," *Proc. IEEE GLOBECOM*, 2020, pp. 1–6.

[11] H. T. Nguyen, T. Van Do, and C. Rotter, "Scaling UPF Instances in 5G/6G Core with Deep Reinforcement Learning," *IEEE Access*, vol. 9, 2021, pp. 165,892–906.

[12] C. Ssengonzi, O. P. Kogeda, and T. O. Olwal, "A Survey of Deep Reinforcement Learning Application in 5G and Beyond Network Slicing and Virtualization," *Array*, vol. 14, 2022, p. 100,142; https://www.sciencedirect.com/science/article/pii/S2590005622000133.

[13] N. Van Huynh *et al.*, "Optimal and Fast Real-Time Resource Slicing with Deep Dueling Neural Networks," *IEEE JSAC*, vol. 37, no. 6, 2019, pp. 1455–70.

[14] Y. Liu, J. Ding, and X. Liu, "A Constrained Reinforcement Learning Based Approach for Network Slicing," *2020 IEEE 28th Int'l. Conf. Network Protocols*, 2020, pp. 1–6.

[15] J. Xiong *et al.*, "Parametrized Deep Q-Networks Learning: Reinforcement Learning with Discrete-Continuous Hybrid Action Space," arXiv:1810.06394 [cs, stat], 2018.

Biographies

Niloy Saha (n6saha@uwaterloo.ca) is a Ph.D. student at the University of Waterloo. He received his Master's degree in computer science from the Indian Institute of Technology, Kharagpur. His research interests are focused on building next-generation mobile networks and intelligent algorithms for their orchestration and management.

Mohammad Zangooei (mzangooei@uwaterloo.ca) is a Ph.D. student at the Computer Science Department of the University of Waterloo. He received his Bachelor's degree in electrical engineering (major) and computer science (minor) from Sharif University of Technology, Tehran, Iran. His research interests revolve around next-generation mobile networks, artificial intelligence, and programmable data planes.

Morteza Golkarifard (mgolkari@uwaterloo.ca) received his B.Sc., M.Sc., and Ph.D. degrees in computer engineering from Sharif University of Technology. He is currently a postdoctoral fellow at the David R. Cheriton School of Computer Science at the University of Waterloo. His research interests include 5G networks, NFV, and SDN.

Raouf Boutaba (rboutaba@uwaterloo.ca) received his M.Sc. and Ph.D. degrees in computer science from Sorbonne University in 1990 and 1994, respectively. He is currently a University Chair Professor and the director of the David R. Cheriton School of Computer Science at the University of Waterloo. He also holds an INRIA International Chair in France. He is the founding Editor-in-Chief of *IEEE Transactions on Network and Service Management* (2007–2010) and the current Editor-in-Chief of *IEEE JSAC*. He is a Fellow of the Engineering Institute of Canada, the Canadian Academy of Engineering, and the Royal Society of Canada.