Fair Virtual Network Function Mapping and Scheduling Using Proximal Policy Optimization

Zhenran Kuai, Tianyu Wang, *Member, IEEE*, and Shaowei Wang, *Senior Member, IEEE*

***Abstract*— Network function virtualization redefines a network service as a softwarized chain of virtual network functions (VNFs), which decouples the specific network service from dedicated devices and greatly reduces the hardware cost. The VNFs are generally deployed on common off-the-shelf servers and statistically share computational resources. Due to the inherent integer constraints, the corresponding VNF mapping and scheduling issue is a highly challenging task. In this paper, we consider the mapping and scheduling of VNFs for a given network service, for which a flexible job shop scheduling problem is formulated to optimize the max-min fairness while ensuring the delay requirements of different service chains. Specifically, we propose a deep reinforcement learning method based on offline proximal policy optimization, which dynamically determines the mapping and scheduling decision based on the state of unfinished service chains. The proposed algorithm is scalable to the number of service chains and can be enhanced by Monte Carlo tree search. Numerical results show that the proposed algorithm outperforms the traditional random forest and the greedy algorithms in terms of both service fairness and acceptance ratio.**

***Index Terms*— Deep reinforcement learning, network function virtualization, proximal policy optimization, virtual network function mapping and scheduling.**

I. Introduction

With the advances of 5G and beyond networks, user experience in a variety of services is constantly emphasized by service providers, who pay more and more attention to service customization so as to meet the quality of service (QoS) requirements of users with limited network resources. The 5G and beyond networks are expected to offer more flexible resource management to satisfy diversified QoS demands [2]. On the other hand, service innovation is accelerating and new services always require a large number of dedicated devices to support diverse network functions. These services demand rapid upgradation and wide deployment of dedicated hardware, incurring high capital expense and operational expenditure [3]. How to support the diverse services in a customizable, flexible, scalable and cost-efficient way is crucial from the viewpoint of the service providers.

Network function virtualization (NFV) provides a promising paradigm for the service providers to deploy and execute network functions, which decouples the network functions from dedicated hardware by using virtual network functions (VNFs) realized in the form of applications on virtual machines or containers [4]. A network service is implemented in the form of a service function chain (SFC) containing a series of particular VNFs according to service semantics. A data flow is traversed through the SFC sequentially to complete the service [5]. When enabling network services with the NFV technology, the dependence on dedicated hardware is weakened and the resources for different services are unified. Along with the NFV, software defined networking decouples the control and the data planes so that the service providers are convenient to manage the SFCs in a centralized manner. As a result, the customizable services can be provided on top of a substrate network, which is constructed by standardized servers and switches.

To provide network services in the NFV framework, several inherent challenges should be addressed. First, all types of required service functions should be deployed on the substrate network and served as VNF instances [6]. For each required VNF type, the number of instances and where to deploy them in the underlying infrastructure should be determined so that the instances can be shared by the same type of VNFs to reduce idle resources [7]. Second, these VNF instances should be assigned and chained together to form complete SFCs, which is known as VNF mapping [8]. The potential sharing of VNF instances across SFCs makes it necessary to schedule the execution sequence of VNFs for a high resource utilization. An effective and efficient scheduling scheme is especially important for the NFV system to improve resource utilization without changing the allocation of hardware resources.

The VNF scheduling has been extensively studied in the literature, where diversified QoS requirements are considered. In [9], reinforcement learning (RL) is proposed to learn the scheduling policy, in which the reward function includes the makespan and the satisfaction of delay. In [10], a reliability-aware VNF scheduling algorithm based on RL is developed, which achieves a trade-off between high reliability and low latency. In [11], auto-scaling VNF instance is introduced during the scheduling process for the considered sharing

Manuscript received 28 December 2021; revised 10 May 2022 and 11 July 2022; accepted 21 September 2022. Date of publication 30 September 2022; date of current version 18 November 2022. This work was partially supported by the National Natural Science Foundation of China under Grants 61931023 and U1936202. An earlier version of this paper was presented at the 2021 IEEE Global Communications Conference, Madrid, Spain, December 2021 [DOI: 10.1109/GLOBECOM46510.2021.9686006]. The associate editor coordinating the review of this article and approving it for publication was C. Li. *(Corresponding author: Shaowei Wang.)*

The authors are with the School of Electronic Science and Engineering, Nanjing University, Nanjing 210023, China (e-mail: dz1923021@smail.nju.edu.cn; tianyu.alex.wang@nju.edu.cn; wangsw@nju.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCOMM.2022.3211071.

Digital Object Identifier 10.1109/TCOMM.2022.3211071

and preemption model, and its effectiveness is verified by a genetic algorithm. In [12], VNF transmission delay is taken into account and also a genetic algorithm is proposed to improve the service acceptance ratio. Since the aforementioned resource allocation steps are coupled, certain decisions can be considered together with the scheduling process to enhance the overall resource utilization of the NFV systems [13], [14], [15], [16]. In [13], an online VNF mapping and scheduling problem is defined, which is solved by greedy strategies and tabu search. In [14], the VNF threading attributes and deployment time in public cloud networks are considered. Since booting a virtual machine or container and installing a VNF instance during deployment are generally time-consuming, a cost-efficient heuristic is proposed to promote the reutilization of existing VNF instances. In [15], the authors tackle VNF mapping, traffic routing and service scheduling jointly while considering bandwidth constraints for the requested services and present a primal-dual decomposition approach to solve the service scheduling problem using column generation technique. In [16], the VNF mapping and traffic routing are investigated to schedule services with stringent deadlines in the context of an ultra-low latency network slice, where a game-theoretic approach is developed by leveraging a decentralized mapping, routing and scheduling process.

In a real NFV system, frequently redeploying VNF instances for continuously arriving SFCs incurs substantial latency inevitably. If the VNF deployment remains unchanged over a large time scale, the latency caused by redeployment can be reduced significantly so that mapping and scheduling can be executed in real time. Meanwhile, the explosion of new applications and mobile terminals in 5G and beyond networks, such as autonomous driving, augmented or virtual reality and unmanned aerial vehicles, etc., are motivating network operators to deliver different live services simultaneously [17]. This implies that the real-time part of resource allocation is necessary for fulfilling diversified QoS requirements. Thus a proper mapping and scheduling across SFCs has become a major challenge for service customization [18], [19], [20]. However, little attention is given to the fairness between services as far as the authors have known.

In this paper, we investigate the VNF mapping and scheduling problem aiming at maximizing the fairness across SFCs while fulfilling service specific end-to-end delay requirements. The optimization task is formulated as a mixed integer program and solved by a deep RL algorithm, where the decisions of mapping and scheduling are incorporated into one action space so that they can interplay efficiently. The policy network takes the features of each mapping and scheduling choice as input, whose size-agnostic property allows the policy network to scale to different problem sizes. In addition, the deterministic property of the mapping and scheduling process is utilized to search for a better decision. We also introduce a Monte Carlo tree search (MCTS) process to improve the decisions produced by the learned policy. The main contributions of this paper are summarized as follows:

- We investigate the joint VNF mapping and scheduling process with the objective of fulfilling specific delay requirements while maintaining fairness. The optimization task is characterized by a Markov decision process (MDP) framework which incorporates the VNF mapping into the scheduling decisions for the the convenience of interplay to achieve a high resource utilization efficiency by optimizing the two decisions simultaneously.
- We propose a deep RL algorithm to learn the mapping and scheduling policy, which is scalable to different numbers of SFCs. Numerical results show that the learned policy yields higher service acceptance ratio and fairness than the traditional random forest and greedy algorithms.
- We incorporate an MCTS into the decision process to improve the policy learned by the deep RL, for which an accurate estimation of the cumulative reward is provided at the cost of the increased computational overhead in the search process. Numerical results show that it can improve the fairness among services with a proper simulation endpoint.

The rest of the paper is organized as follows. In Section II, we give system model and formulate the VNF mapping and scheduling problem. In Section III, we propose a deep RL method to learn the mapping and scheduling policy. In Section IV, we incorporate MCTS into the mapping and scheduling process to improve the learned policy. Numerical results are analyzed in Section V and conclusions are drawn in Section VI.

II. System Model and Problem Formulation

A. Model Description

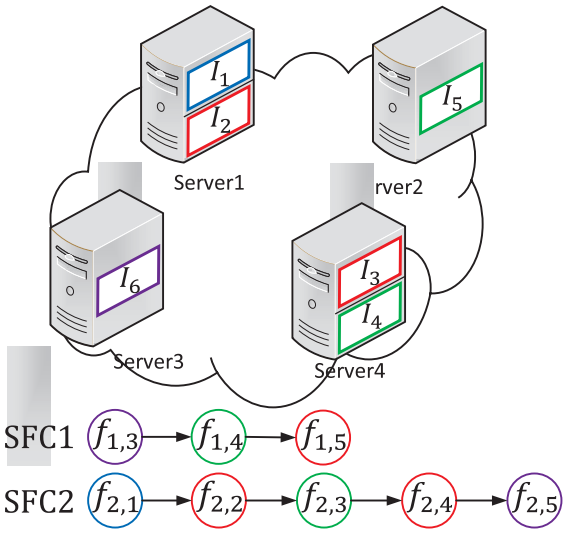
Consider a substrate network consisting of multiple commodity servers. K VNF instances are already deployed on the servers, the set of which is denoted by $\mathcal{I} = \{1, \ldots, K\}$. Each instance k is the realization of a specific VNF type $T_k \in \mathcal{T}$, where $\mathcal{T} = \{1, \ldots, D\}$ contains the types of all VNF instances. The time horizon is slotted and we focus on a certain slot. We denote by $\mathcal{N}_u = \{1, \ldots, N_u\}$ the set of the SFCs that are not completed in previous slots, and by $\mathcal{N}_a = \{1, \ldots, N_a\}$ the set of the new arrived SFCs in the current slot. Then, the set of all SFCs is denoted by $\mathcal{N} = \mathcal{N}_u \cup \mathcal{N}_a$.

For each SFC $i \in \mathcal{N}$, the set of VNFs in i is denoted by $\mathcal{F}_i$. Completing SFC i requires executing $|\mathcal{F}_i|$ VNFs in a specific order denoted as $f_{i1} \rightarrow \cdots \rightarrow f_{i|\mathcal{F}_i|}$, where the element f_{ij} is the j-th VNF in SFC i. If SFC $i \in \mathcal{N}_u$, partial VNFs should have been completed, so we specify that $\mathcal{F}_i$ only includes the VNFs that are uncompleted. The number of all considered VNFs is $N_f = \sum_i |\mathcal{F}_i|$. For each f_{ij}, the set of the same type of VNF instances is denoted by $\mathcal{I}_{ij} = \{k \in \mathcal{I} \mid T_k = T(f_{ij})\}$, where $T(f_{ij})$ indicates the VNF type of f_{ij}. If f_{ij} is executed by $k \in \mathcal{I}_{ij}$, its processing delay is calculated as $\rho_{ijk} = \omega_i / p(I_k)$, where ω_i is the traffic size of SFC i and $p(I_k)$ is the process rate of instance k. Denote by t_i^d the deadline of SFC i. We consider the scenario with VNF instances deployed in a local data center, where the network is assumed to be fully connected and the transmission delay is much shorter than the processing delay. Thus, the transmission delay can be ignored. Note that many existing works also consider the scenario without transmission delay,

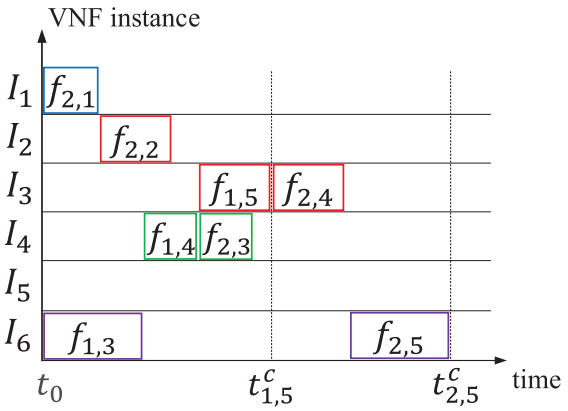
where the main limitation of network resource is considered to be computation capacity instead of network transmission [10], [11], [13].

We should first decide on which instance to map each VNF before completing an SFC. We denote by m_{ijk} as the mapping decision, which equals to 1 if f_{ij} is executed by $k \in \mathcal{I}_{ij}$ or 0 otherwise. For SFC $i \in \mathcal{N}_u$, all VNFs have been mapped in previous slots, so the mapping results are reserved to reduce the computational cost and delay caused by remapping. VNFs should be executed sequentially along the SFC to complete the service once they are already mapped. One instance can process at most one VNF at a time. If multiple VNFs in different SFCs are mapped to the same VNF instance, their execution orders should be scheduled. Here, we directly use the start processing time of f_{ij} to represent the scheduling decision of one VNF, which is denoted by t^s_{ij}. The completion time of f_{ij} is calculated as $t^c_{ij} = t^s_{ij} + \sum_k m_{ijk}\rho_{ijk}$. A complete schedule is obtained when the mapping choices and start time of all VNFs are determined. The SFC is rejected by the system if the completion time is greater than the deadline.

An example of the VNF mapping and scheduling is illustrated in Fig. 1. Fig. 1(a) shows a substrate network of 4 servers with 6 deployed VNF instances. The k-th instance is represented by I_k. The instances depicted in the same color belong to the same type, such as I_2 and I_3. We consider the mapping and scheduling of two SFCs in a slot t. The first SFC is an uncompleted SFC whose first two VNFs, $f_{1,1}$ and $f_{1,2}$, have been executed. $f_{1,3}$, $f_{1,4}$ and $f_{1,5}$ have been mapped to I_6, I_4 and I_3, respectively. The second SFC is a new arrived SFC. The choices of mapping for each VNF can only be the instances of the same color. Given that $f_{2,2}$ is mapped to I_2 and $f_{2,4}$ is mapped to I_3, the execution orders of $f_{1,5}$ and $f_{2,4}$ should be scheduled. If the first SFC is more delay-sensitive, $f_{1,5}$ might be scheduled prior to $f_{2,4}$. The result can be drawn as a Gantt chart after all the VNFs are mapped and scheduled, as shown in Fig. 1(b).



(a) Deployed VNF instances and SFCs.



(b) Gantt chart of a mapping and scheduling case.

Fig. 1. Illustration of VNF mapping and scheduling.

B. Constraints

The following constraints are imposed to guarantee the feasibility of the solutions.

1) Mapping Constraints: The mapping results of the SFCs in $\mathcal{N}_u$ are reserved. This mapping constraint is expressed by

$$m_{ijk} = \mathbb{I}(k = I^u_{ij}), \quad \forall i \in \mathcal{N}_u, j \in \mathcal{F}_i, k \in \mathcal{I}_{ij}, \tag{1}$$

where I^u_{ij} represents the instance to which f_{ij} in SFC $i \in \mathcal{N}_u$ is mapped. $\mathbb{I}(\cdot)$ represents an indicator function that equals 1 if an event holds or 0 otherwise. For each SFC $i \in \mathcal{N}_a$, one VNF can only be mapped to one instance, thus we have

$$\sum_{k \in \mathcal{I}_{ij}} m_{ijk} = 1, \quad \forall i \in \mathcal{N}_a, j \in \mathcal{F}_i. \tag{2}$$

2) Precedence Constraints: Each data flow should traverse through the VNFs sequentially to complete the service, which means that the traffic should be traversed to the successor VNF after being processed by the predecessor VNF along an SFC. The precedence constraint is expressed by

$$t^s_{i,j+1} - t^s_{ij} \geq \sum_{k \in \mathcal{I}_{ij}} m_{ijk}\rho_{ijk}, \quad \forall i \in \mathcal{N}, j \in \mathcal{F}_i \setminus \{|\mathcal{F}_i|\}, k \in \mathcal{I}_{ij}. \tag{3}$$

3) Resource Constraints: If multiple VNFs in different SFCs are mapped to the same instance, their execution orders should be orchestrated since an instance can process at most one VNF at a time. The resource constraints are given by

$$\mathbb{I}(t^s_{ij} > t^s_{i'j'})(t^s_{ij} - t^s_{i'j'}) \geq \mathbb{I}(t^s_{ij} > t^s_{i'j'})m_{ijk}m_{i'j'k}\rho_{i'j'k}, \quad \forall i, i' \in \mathcal{N} : i < i', j \in \mathcal{F}_i, j' \in \mathcal{F}_{i'}, k \in \mathcal{I}_{ij} \cap \mathcal{I}_{i'j'}, \tag{4}$$

$$\mathbb{I}(t^s_{i'j'} > t^s_{ij})(t^s_{i'j'} - t^s_{ij}) \geq \mathbb{I}(t^s_{i'j'} > t^s_{ij})m_{ijk}m_{i'j'k}\rho_{ijk}, \quad \forall i, i' \in \mathcal{N} : i < i', j \in \mathcal{F}_i, j' \in \mathcal{F}_{i'}, k \in \mathcal{I}_{ij} \cap \mathcal{I}_{i'j'}. \tag{5}$$

4) Variable-Type Constraints: The variable-type constraints are given by

$$t_{ij}^{s} \geq 0, \qquad \forall i \in \mathcal{N}, j \in \mathcal{F}_i, \tag{6}$$

$$m_{ijk} \in \{0,1\}, \quad \forall i \in \mathcal{N}_a, j \in \mathcal{F}_i, k \in \mathcal{I}_{ij}. \tag{7}$$

C. Problem Analysis

The objective of each SFC is to satisfy its individual end-to-end delay requirements. We define the earliness of SFC i as $t_i^d - t_{i|\mathcal{F}_i|}^c$, which is the time advance of its completion over the deadline. The earliness can be formally expressed as

$$E_i = t_i^d - (t_{i|\mathcal{F}_i|}^s + \sum_{k \in \mathcal{I}_{i|\mathcal{F}_i|}} m_{i|\mathcal{F}_i|k} \rho_{i|\mathcal{F}_i|k}). \tag{8}$$

For each SFC, the goal is to maximize its earliness to guarantee the satisfaction of the delay requirement. However, maximizing the earliness of one SFC may defer the completion of other SFCs that share the same VNF instances. The unfairness may bring more deadline violations. Therefore, we maximize the minimum earliness over all SFCs to prevent one SFC from having significant superiority over the others. With the constraints in (1)-(7), the whole problem can be formulated as follows.

$$\max_{m_{ijk}, t_{ij}^s} \min_i \{t_i^d - (t_{i|\mathcal{F}_i|}^s + \sum_{k \in \mathcal{I}_{i|\mathcal{F}_i|}} m_{i|\mathcal{F}_i|k} \rho_{i|\mathcal{F}_i|k})\},$$
$$s.t.\ (1) - (7). \tag{9}$$

The VNF mapping and scheduling problem formulated in (9) falls into the category of job-shop scheduling problems, where multiple jobs are scheduled on a set of machines concurrently. A job consists of several ordered operations, each of which can be processed on a specific machine and occupies a period of time. If each operation has a set of available machines to be processed on, it becomes a flexible job-shop scheduling problem, which is exactly what our problem belongs to.

The flexible job-shop scheduling problem is NP-hard [21] and has been investigated extensively. Many existing methods can generate effective solutions in small scale problems but are not applicable when the size of problem scales up. Priority dispatching rule (PDR) [22] is a heuristic method that has been widely used in real-world scheduling systems to generate a feasible schedule efficiently. In the context of our problem, the PDR iteratively builds up a complete map and schedule by determining the mapping and scheduling decisions of a VNF among the first undetermined VNFs in all SFCs. Specifically, mapping and scheduling priority functions are used to rank the mapping and scheduling decisions respectively for all involved VNFs. The VNF with the highest scheduling priority is scheduled first and mapped to the instance with the highest mapping priority.

The PDR is computationally fast and easy to implement. However, the following factors restrict its design for our problem: 1) designing effective PDRs requires substantial engineering experience and trial-and-error to tune the rules or parameters of the priority functions; 2) a PDR is often designed for a specific scenario and the tedious design process has to be restarted for new scenarios; and 3) designing individual PDRs for mapping and scheduling subproblems cannot take the existing interplay into consideration [23]. Therefore, we adopt deep RL to design the PDR for the VNF mapping and scheduling problem automatically to deal with the above restrictions.

III. Deep Reinforcement Learning for VNF Mapping and Scheduling

We first describe the VNF mapping and scheduling process under an MDP framework, then present the deep RL algorithm in detail.

A. Markov Decision Process Formulation

We follow a similar process as PDR to perform mapping and scheduling. In each step, the first VNF that is not mapped and scheduled yet in each SFC is included in an available VNF set. The mapping and scheduling choices of the VNFs in the set construct the decision space. We select a VNF from the set and determine its mapping and scheduling scheme. The start time of the selected VNF is arranged on the settled instance as early as possible while satisfying the constraints in (3)-(5) to generate a tight schedule [24]. Then, the successor VNF along the SFC becomes the first undetermined VNF and joins the available VNF set. The mapping decisions and start time of all VNFs are determinate after N_f consecutive steps. The main components in the MDP are illustrated as follows.

1) Action: An action in our problem is a mapping and scheduling scheme for a VNF. If we make decisions for all SFCs in a map-then-schedule style and optimize these two processes separately, the neglect of the existing interplay will lead to inefficient resource utilization [15]. Therefore, we consider integrating the VNF mapping and scheduling decisions into one action space. As shown in Fig. 2, suppose $f_{1,3}$ and $f_{2,1}$ have been mapped and scheduled, $f_{1,4}$ and $f_{2,2}$ are in the available set. $f_{2,2}$ can be mapped to I_2 or I_3. $f_{2,2}$ or $f_{1,4}$ can be selected for scheduling. The mapping choices of $f_{2,2}$ can be incorporated into the scheduling of $f_{1,4}$ and $f_{2,2}$. Then, we have three options for scheduling: $f_{1,4}$ on I_4, $f_{2,2}$ on I_2 and $f_{2,2}$ on I_3, which are called candidate actions. The mapping and scheduling decisions of the involved VNF are determined after executing one of these actions. Let $\mathcal{A}_n$ represent the set of candidate actions in step n. Notice that $|\mathcal{A}_n|$ is variable because the lengths of SFC are usually different and so is the number of available instances for each VNF.

2) State: A state $s_n \in \mathcal{S}$ in our problem must represent both the mapping and scheduling progresses. We use the raw features of all candidate actions in step n to represent the status of all SFCs. The state is defined as $s_n = [h_n^1, \ldots, h_n^{|\mathcal{A}_n|}]$, where h_n^k refers to the raw features of the k-th action in $\mathcal{A}_n$. The notations of all features are listed in Table I. Note that these features are either derived from the SFC where the action is located or involved with the related VNF instances. For candidate actions, t_{ijk}^{LB} is calculated as $t_{ijk}^{LB} = t_{i,j-1}^c + \rho_{ijk}$, while t_{ijk}^c is calculated based on the precedence and resource constraints in (3)-(5). In particular, t_{ijk}^{LB} and t_{ijk}^c are

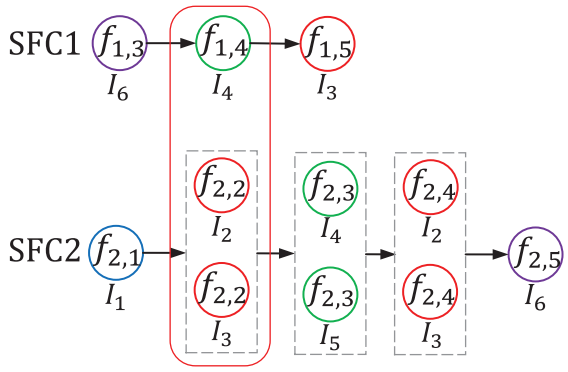


Fig. 2. Candidate actions.

determined for VNFs that have been mapped and scheduled. The slack time t_i^{SL} is calculated as $t_i^{SL} = t_i^d - t_{ijk}^{LB}$, which implies the spare time before the deadline. For an SFC $i \in \mathcal{N}_u$, if the j-th VNF f_{ij} mapped to I_k is selected, t_k^r is updated as $t_k^r = t_k^r - \rho_{ijk}$. We denote by $\mathcal{F}_k^U$ the set of the same type of VNFs as I_k that have not been mapped and scheduled before n. t_k^U is calculated as $t_k^U = \sum_{f_{ij} \in \mathcal{F}_k^U} \rho_{ijk}/|\mathcal{I}_{ij}|$.

3) State Transition: Once an action is executed, the actions relating to the same VNF are excluded from the set of candidate actions. The actions relating to the successor VNF in the same SFC are included as candidate actions for the next step $n+1$. Features of the actions in $\mathcal{A}_{n+1}$ constitute s_{n+1}.

4) Reward: The minimum earliness can only be observed after a complete schedule is generated. Treating this objective as a reward function is difficult to train the RL agent, since the reward is too sparse and uninformative to learn about the action. To mitigate the impact of reward sparsity, the minimum earliness is decomposed into reward components:

$$r(s_n, a_n) = \min_{i,j}\{t_i^d - t_{ij}^c(n+1)\} - \min_{i,j}\{t_i^d - t_{ij}^c(n)\}, \quad (10)$$

where $t_{ij}^c(n)$ denotes the completion time of the mapped and scheduled VNF f_{ij} before step n. Eq. (10) calculates the decrement of minimum earliness caused by a_n in s_n. The cumulative discounted reward is calculated as $\sum_{n=1}^{N_f} \gamma^{n-1} r(s_n, a_n)$. If the discount factor $\gamma = 1$, the cumulative reward is equal to $\min_{i,j}\{t_i^d - t_{ij}^c\} - \min_i\{t_i^d\}$, which conforms to the minimum earliness. Optimizing the policy with respect to the cumulative reward will incline the RL agent to give up the actions with the best current reward to get a better long-term performance.

5) Policy: In step n, the policy function $\pi_\theta : \mathcal{S} \to \mathcal{P}(\mathcal{A}_n)$ takes the state s_n as input and outputs a distribution over all actions in $\mathcal{A}_n$.

B. Proximal Policy Optimization

The goal of an RL agent is to learn a high-quality policy by interacting with environment. To this end, the agent needs to go through different states and actions to evaluate and optimize the policy iteratively. We implement this process by using proximal policy optimization (PPO) [25].

TABLE I
NOTATIONS TO DESCRIBE A STATE

Notation	Description
b_i	Indicate whether the SFC i is completed.
e_k^T	The time efficiency of the VNF instance k to process the mapped VNF.
t_i^d	The deadline of the SFC i.
t_i^{SL}	The slack time of the SFC i before the deadline.
t_{ijk}^c	The completion time of f_{ij} on instance k.
t_{ijk}^{LB}	The lower bound of the completion of f_{ij} on instance k.
t_k^r	The total time required for the VNF instance k to process the VNFs of SFCs in $\mathcal{N}_u$.
t_k^A	The total time required for the VNF instance k to process the mapped VNF.
t_k^U	The expected time for instance k to process the VNFs that have not been mapped.
ρ_{ijk}	The processing time of f_{ij} on instance k.

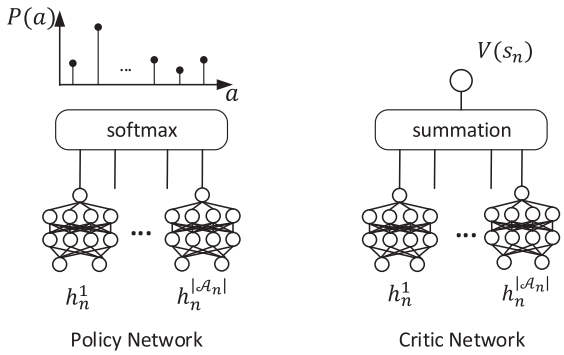


Fig. 3. Illustration of the policy and critic networks.

1) Policy Network: In the VNF mapping and scheduling problem, the policy π_θ is optimized to generate an optimal distribution over the action space $\mathcal{A}_n$ in s_n. A neural network is used as a policy function approximator to deal with the large and continuous state space. However, it is difficult for a neural network of fixed architecture to map the state space to the distributions over the action space, because the action space is variable in the decision process. A specific structure of the network should be devised to accommodate the changing size of the action space. Specifically, we adopt a multi-layer perceptron to obtain a scalar value $v(a_n^k) = MLP_\theta(h_n^k)$ for each action a_n^k in $\mathcal{A}_n$. MLP_θ can be regarded as the priority function of the PDR and the scalar v can be regarded as the priority value for the candidate action. A softmax operation is performed over the scalar values to output a distribution $P(\mathcal{A}_n)$, from which the action a_n is sampled. As shown in Fig. 3, the parameters of the policy network are shared across all candidate actions and the size of MLP_θ has nothing to do with that of the action space. Therefore, the policy network can deal with the variable action space and is potentially size-agnostic for different numbers of SFCs.

2) Critic Network: Once action a_n is executed, r_n and s_{n+1} will be observed. Repeating this process will generate a sampled action trajectory τ of length N_f: $\{(s_1, a_1, r_1), \ldots, (s_{N_f}, a_{N_f}, r_{N_f})\}$, which can be utilized to

Algorithm 1 PPO-Based VNF Mapping and Scheduling

Initialize the policy network $\pi_\theta(a|s)$ with θ.
Initialize the critic network $V_\phi(s)$ with ϕ.
for iteration $t = 1, 2, \cdots, N_t$ **do**
  for environment $e = 1, 2, \cdots, N_e$ **do**
    Take actions according to the policy π_θ to generate a trajectory τ_e in the e-th environment;
  end for
  $\theta_{old} \leftarrow \theta$;
  for epoch $\delta = 1, 2, \cdots, N_{ep}$ **do**
    Compute $r_i(\theta)$ and $\hat{A}_i$ with τ_e for each environment;
    Compute $L_e^F(\theta, \phi)$ for each environment;
    Update θ and ϕ by a gradient method w.r.t $(\sum_e L_e^F(\theta, \phi))/N_e$;
  end for
end for

evaluate and optimize π_θ. Although the policy can be simply evaluated with the discounted cumulative reward $\sum_n \gamma^{n-1} r_n$, the variance is too large and the parameters of the policy will be updated in the wrong direction with a high probability. Thus the policy is evaluated with a critic network $V_\phi : \mathcal{S} \rightarrow \mathbb{R}$, which estimates the expected cumulative reward in state s_n to reduce the variance. In our design, we adopt the same structure as the policy network, except that the last operation is the summation over the scalar values, as shown in Fig. 3. The critic network is updated to minimize the mean square error between $V_\phi(s_n)$ and the target cumulative reward:

$$L^{VF}(\phi) = \sum_{n=1}^{N_f} \left[\sum_{i=n}^{N_f} \gamma^{i-1} r_i - V_\phi(s_n) \right]^2. \tag{11}$$

Note that the number of outputs in each step may not be constant due to the variable action space. Thus, taking a summation may not be suitable for the estimation of the cumulative reward. We will set the number of available instances for each VNF type to be the same in the learning phase, which fixes the size of the action space for convenience.

3) Policy Optimization: Traditionally, a gradient ascent algorithm is applied on a sample of the performance objective $J(\theta)$ to optimize the policy [26]. The performance objective $J(\theta)$ and its gradient are written by

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[\sum_{n=1}^{N_f} \gamma^{n-1} r_n \right], \tag{12}$$

$$\nabla J(\theta) \approx \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[\sum_{n=1}^{N_f} \nabla_\theta \log \pi_\theta(a_n|s_n) \hat{A}_n \right], \tag{13}$$

where $\hat{A}_n = r_n + \gamma V_\phi(s_{n+1}) - V_\phi(s_n)$ measures the advantage of a_n over the other actions in s_n. The objective is the cumulative discounted reward calculated based on the sampled action trajectory τ.

Although there are other RL methods to learn the policy based on Eqs. (12) and (13), we adopt the PPO framework in this paper. First, the PPO can address the discrete and variable action space which is the case of our considered mapping and scheduling problem. Second, most existing RL methods do not explore whether the policy will be improved after a gradient descent. The PPO tries to guarantee the improvement of the updated policy over the old policy by applying importance sampling to the objective, which is expressed by

$$L^{CLIP}(\theta) = \sum_{n=1}^{N_f} \left[\min(\omega_n(\theta)\hat{A}_n, \text{clip}(\omega_n(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_n) \right], \tag{14}$$

where $\omega_n(\theta) = \frac{\pi_\theta(a_n|s_n)}{\pi_{\theta_{old}}(a_n|s_n)}$. The first term inside the min, $\omega_n(\theta)\hat{A}_n$, exploits the samples of an old policy $\pi_{\theta_{old}}$ to approximate the performance objective of the updated policy π_θ. π_θ is updated iteratively by minimizing $\omega_n(\theta)\hat{A}_n$ to guarantee its improvement over $\pi_{\theta_{old}}$. However, if the divergence between $\pi_{\theta_{old}}$ and $\pi_{\theta_{old}}$ is excessively large, approximating the objective of π_θ by using importance sampling will result in a large bias [25]. Thus, $\omega_n(\theta)$ is clipped into the interval $[1-\epsilon, 1+\epsilon]$ to limit the probability ratio, which improves the stability of the optimization process.

The aforementioned objectives can be combined as

$$L^F(\theta, \phi) = c_1 L^{CLIP}(\theta) - c_2 L^{VF}(\phi) + c_3 L^E(\pi_\theta(\cdot|s_n)), \tag{15}$$

where c_1, c_2 and c_3 are coefficients. L^E represents an entropy bonus that encourages to explore actions. The parameters of the policy and critic networks are updated to minimize $L^F(\theta, \phi)$ with a gradient ascent step iteratively. Each gradient is averaged over the sampled action trajectories of multiple parallel environments to reduce the variance. The proposed PPO-based mapping and scheduling procedure is summarized in Algorithm 1.

Note that the learned policy can be regarded as the priority function of the PDR for the VNF mapping and scheduling problem. In each step, we need to execute one action in the action space. The priority values of the candidate actions are calculated by MLP_θ, whose input is only for a single action. This ensures that the size of the network has nothing to do with that of the action space. The action with the largest priority value is selected for mapping and scheduling. A complete schedule is generated by iteratively executing actions for all VNFs. In this process, the computation load of determining the mapping and scheduling decisions for each VNF only grows linearly with the size of action space, which guarantees the scalability of the method.

IV. MCTS-Based Planning

In the VNF mapping and scheduling problem, the occupancy of all VNF instances and the scheduling progress of all SFCs are definite if a mapping and scheduling decision of a VNF is taken. This indicates the state transition in our MDP formulation is deterministic and s_{n+1} is known to the agent once an action a_n is selected in s_n. Therefore, we could know the cumulative reward of a series of hypothetical future actions, which contains more information than the defined state and should be exploited to improve the performance

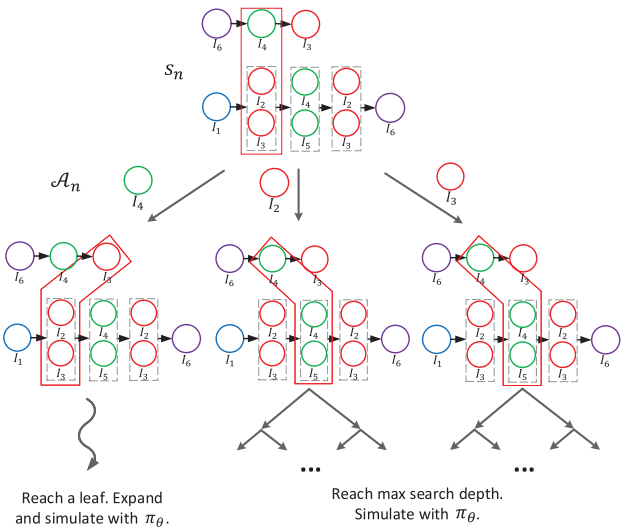


Fig. 4. The MCTS for VNF mapping and scheduling.

Algorithm 2 PPO-MCTS Algorithm

Input: Policy π, root state s_n, simulation endpoint h, max search depth d, the number of iterations q.
Output: Action a_n.
Build a search tree with a root node $g = s_n$.
for iteration $t = 1, 2, \cdots, q$ **do**
  while g is not a leaf or d is not reached **do**
    Select a child node g_c according to UCT;
    Expand if g_c is a leaf and d is not reached;
    Simulate with π to h if g_c is a leaf;
    Backpropagate with max reward if g_c is a leaf;
    $g \leftarrow g_c$;
  end while
end for
$a_n \leftarrow$ The action that produces maximum return in s_n;
return a_n

of the learned policy. We choose to search for a better a_n by exploring the sequences of future actions, which is also referred to as planning in RL. Assume that all types of VNF have the same number of instances deployed on the substrate network(i.e., $|\mathcal{I}_{ij}| = \mu$, $\forall f_{ij}$). Enumerating all the sequences of actions for d lookahead steps becomes prohibitive with an $O((N_u + \mu N_a)^d)$ complexity. Thus, we adopt the MCTS to enable an efficient search [27].

The basic idea of the MCTS involves searching for an optimal decision by iteratively building a tree from the root node until reaching a predetermined computational budget, e.g., maximum iterations. After a round of search, the best child of the root node is returned and serves as the root node in the next round. We perform the MCTS to output an action a_n in s_n. Each node represents a state and each directed link to a node represents the corresponding action that leads to the state, as shown in Fig. 4. The search process consists of four processes, i.e., selection, expansion, evaluation, and backpropagation.

In the selection step, upper confidence bound applied to trees (UCT) is used for search [28]. We start from the root node s_n and select the child node with the largest UCT value, which is expressed by

$$UCT(g) = Q(g) + c_u \frac{\sqrt{N_v(g_f)}}{1 + N_v(g)}, \tag{16}$$

where c_u is a tunable weight determining the level of exploration. $Q(g)$ is the estimated cumulative reward in node g, and N_v is the corresponding visit count. g_f is the father node of g. Selecting the node with the largest UCT value is to explore the nodes that are rarely visited while exploiting the nodes with a large cumulative reward. The selection continues until it reaches a leaf or the predefined maximum search depth d.

In the expansion step, a leaf node is expanded if a selection reaches a leaf but not the maximum depth. A random child node of the selected node is added to the tree.

In the evaluation step, the cumulative reward of a leaf should be estimated as the search process reaches it. Although the role of a critic network in deep RL is exactly to estimate the cumulative reward of a state [29], the critic network in the proposed method cannot be straightforwardly used for the evaluation in the MCTS. First, the rewards of the learning phase are batch normalized to facilitate the fitting process of the critic network, whose output cannot be calculated together with the original reward to evaluate a node. Second, the summation operation is adopted for the convenience of learning and is not suitable for estimating the cumulative reward in the scenario of variable action space. These prevents the critic network from application. In the MCTS procedure, we execute the learned policy at the leaf node to obtain the cumulative reward. However, executing the policy to the end of the task is costly. We set the predefined execution endpoint h for the policy execution, which ensures that the nodes at the same depth are evaluated based on the same number of hypothetical future actions.

In the backpropagation step, the cumulative reward and the visit count of the nodes selected in this round of search are updated. The visit count of each node is incremented. The maximum cumulative reward is backpropagated in our procedure instead of the mean value in [29], since the goal is to find the best sequence of actions of the next d steps to optimize the current decision with the learned policy.

As the predefined number of searches is reached, the child node with the largest Q is returned. We execute only the first action a_n that leads to the maximum Q in s_{n+1} and start another planning process for s_{n+1}. If the action with the highest priority value is within the set of actions that produce the maximum reward, then it will be selected to guarantee that the planning is as good as the learned policy. The designed planning procedure allows a controllable search for the scheduling agent with a complexity of $O(hq)$, where q is the number of iterations in determining an action with the MCTS. Therefore, the computational overhead of the planning algorithm is hq times higher than that of the learned policy. The MCTS-based planning (PPO-MCTS) procedure is summarized in Algorithm 2.

TABLE II
SIMULATION PARAMETERS

Parameter	Small-scale	Large-scale
Number of Servers	5	5
Types of VNF instances	5	10
μ	$\{2, 3, 4\}$	2
Number of SFCs	$[4, 12]$	$[16, 24]$
Length of SFCs	$[4, 10]$	$[4, 10]$
Traffic size of each SFC	$[1, 5]$ Mbits	$[5, 10]$ Mbits
Deadline of SFC	$[1.2\eta, 2.5\eta]$	$[1.2\eta, 2.5\eta]$
Slot length	1 ms	1 ms
Process rate of VNF instance	$\{1, 2, 3\}$ Gbps	$\{1, 2, 3\}$ Gbps

TABLE III
HYPERPARAMETERS IN THE LEARNING PROCESS

Parameter	Value
γ	1
ϵ	0.1
Coefficient of L^{CLIP}	2
Coefficient of L^{VF}	0.05
Coefficient of L^{E}	0.01
Learning rate	0.0002
Optimizer	Adam
Number of iterations	$N_t = 10000$
Number of parallel environments	$N_e = 100$
Number of epochs	$N_{ep} = 3$
Activation function	ReLU

V. NUMERICAL RESULTS

We compare the PPO algorithm with the other four methods: column generation (CG) [15], random forest (RF) [30], greedy best availability (GBA) [13] and random scheduling (RS) algorithms. The PPO-MCTS is compared with the PPO and the Google Or-Tools (ORT) [31] to see how much it can improve the learned policy.

A. Settings

1) Baseline Algorithms: In the CG algorithm, CG technique is applied to solve the relaxed MILP formulation of the SFC scheduling problem and obtain the linear program solution. The integer linear program (ILP) solution is obtained by removing the relaxation in the formulation. Note that the objective in [15] is minimizing the time needed to complete the processing of all the services, i.e., makespan. We hence change the optimization objective to maximize the minimum earliness while keeping the MILP formulation to make the comparisons more reasonable. In the RF-based algorithm, RF is used to extract the PDRs of mapping and scheduling from the solutions generated by ORT. The GBA scheduling algorithm greedily selects the VNF instance with the earliest completion time of the current VNF queue for mapping and scheduling. The RS algorithm also follows the rule of PDR but selects a random action from the available action set in each step.

2) Simulation Setup: Consider an NFV system of 5 servers. We compare the performance of the considered methods over small-scale scenarios and large-scale scenarios. In small-scale scenarios, five types of VNF instances are deployed on the network. For each type, the number of available instances for mapping is set to the same value μ. The process rate of each VNF instance is randomly chosen from $\{1, 2, 3\}$ Gbps with equal probability.

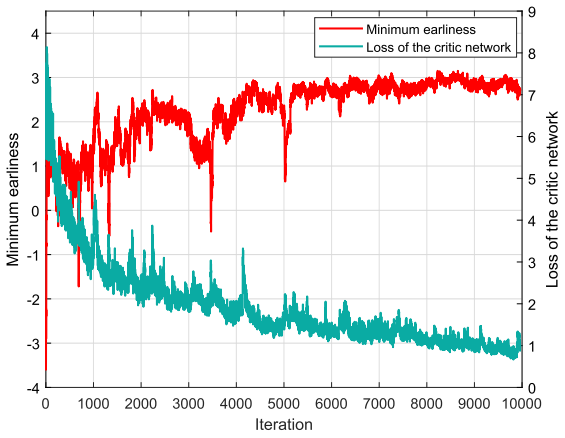


Fig. 5. Average minimum earliness and loss of the critic networks in the learning process of the PPO algorithm. The number of parallel environments is $N_e = 100$.

For network services, the number of unfinished SFCs is $N_u = 4$ and the number of arrived SFCs is equal to N_u. The length of arrived SFCs is $|\mathcal{F}_i| = 6$ and the length of unfinished SFCs is half that of the arrived SFCs. Each VNF in SFCs is randomly chosen from all types with equal probability. The mapping result of each VNF in an unfinished SFC is randomly chosen from the available instances with equal probability. The traffic size of each SFC is set to an integer that is uniformly distributed over $[1, 5]$ Mbits. The time slot length is 1 ms. Let η represent the total processing delay of the VNFs belonging to an SFC. The deadline of the SFC is uniformly distributed between $[1.2\eta, 2.5\eta]$. These parameters are generated independently in each environment in the learning and testing phases. For large-scale scenarios, the parameters are generated in the same way, but with different ranges. The simulation parameters are summarized in Table II.

The policy and critic networks share all layers that precede the last hidden layer to accelerate learning. The hyperparameters of the learning process are summarized in Table III. All the numerical experiments are executed on a computer with Intel(R) Core(TM) i7-9700 CPU @ 3.00 GHz and an Nvidia GeForce RTX 2060 GPU.

B. Performance of the PPO Algorithm

First, the policy network should be trained offline. Since the solution space grows exponentially with the flexibility of VNF mapping and the number of SFCs, we train a policy network on small-scale problems to reduce the difficulty of learning, where the numbers of unfinished and arrived SFCs are both set to 4 and the number of available instances for mapping is set to 2.

Fig. 5 shows the average minimum earliness and loss of the critic networks in the learning process of the PPO algorithm. As we can see, the average minimum earliness increases with the iteration process. This implies that the reward sparsity is well resolved by the designed reward decomposition and the priority values of mapping and scheduling decisions are

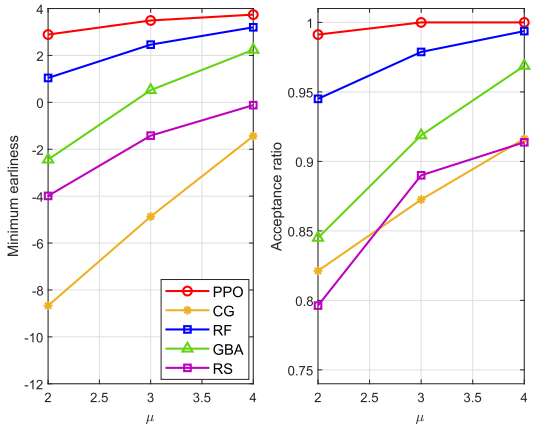


Fig. 6. Average minimum earliness and acceptance ratio as a function of the number of available instances that each VNF can be mapped. $N_u = 4$, $N_a = 4$ and $|\mathcal{F}_i| = 6, \forall S_i \in \mathcal{Q}$.

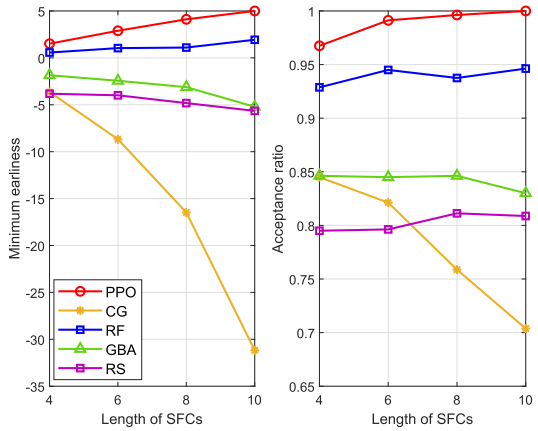


Fig. 7. Average minimum earliness and acceptance ratio as a function of the length of SFCs. $N_u = N_a = 4$ and $\mu = 2$.

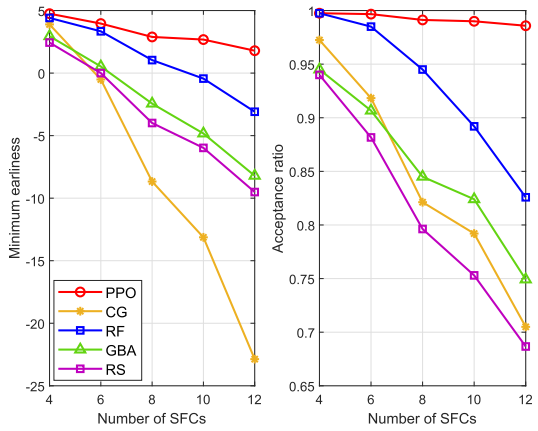


Fig. 8. Average minimum earliness and acceptance ratio as a function of the number of SFCs. $\mu = 2$ and $|\mathcal{F}_i| = 6, \forall i \in \mathcal{N}_a$.

learned with the policy network. Eventually, the minimum earliness converges to a value greater than zero, which indicates that most of the end-to-end delay requirements are satisfied and all SFCs are completed with nearly equal earliness. Moreover, the convergence of the loss proves that the summation operation is suitable to construct the critic network for the training phase.

Since the policy network is designed to be size-agnostic, we compare the minimum earliness and acceptance ratio of the learned policy with other methods in different settings to demonstrate its superiority and scalability. The acceptance ratio is defined as the proportion of the total SFCs that are completed before deadlines. The result for each setting is averaged over 100 random environments.

Fig. 6 shows the average minimum earliness and acceptance ratio as a function of the number of available instances that each VNF can be mapped, i.e., μ. It can be seen that the proposed PPO algorithm outperforms other methods, which indicates that the policy learned by PPO can generate an effective schedule on SFCs. However, the PPO shows a decreasing advantage over other methods with the increasing flexibility of VNF mapping. The reason is that the VNF instances are plentiful such that there is no need for the same type of VNF to compete for one instance. The advantage brought by efficient resource utilization diminishes.

Fig. 7 shows the average minimum earliness and acceptance ratio as a function of the length of SFCs. As the length of SFCs increases, the minimum earliness of the PPO algorithm increases and its advantage over other algorithms becomes greater. This is because the increase in the length of SFCs improves the probability that SFCs share common VNFs. The learned policy can effectively reduce the delay caused by resource competition, which widens the performance gap. As for the acceptance ratio, there is no marked widening of the gap since the delay requirement also increases with the length of SFCs in our settings. The performance of CG algorithm drops with increasing length of SFCs since the CG algorithm relaxes the integrality of the problem and finds an ILP solution by removing the relaxation. Therefore, a certain gap exists between the ILP solution and the optimal solution of the relaxed MILP formulation. The larger the problem size, the more variables of the formulation, resulting in a larger gap these two solutions.

Fig. 8 shows the average minimum earliness and acceptance ratio as a function of the number of SFCs. Both the minimum earliness and acceptance ratio decrease as the number of SFCs increases. The reason is that more VNFs have to wait for the completion of other VNFs when competing for the same VNF instance. The performance gaps between the PPO algorithm and the others also expand as the increasing of the SFCs, which implies that our proposed algorithm can maintain a much better performance than the others even if the system is heavy load.

Fig. 9 shows the performance of fairness in different settings. We define the standard deviation of earliness (STDE) as a new metric to focus only on fairness. Specifically, a small STDE means that all services are completed with nearly equal earliness, implying a fair mapping and scheduling result. As can be seen, the PPO algorithm shows the lowest STDE in most cases. The STDE of the CG algorithm increases the fastest as the number or length of SFCs increases. The reason is that removing the relaxation to obtain the ILP solution in the CG algorithm leads to a larger discrepancy with the optimal solution and the ILP solution is not a tight schedule. As the

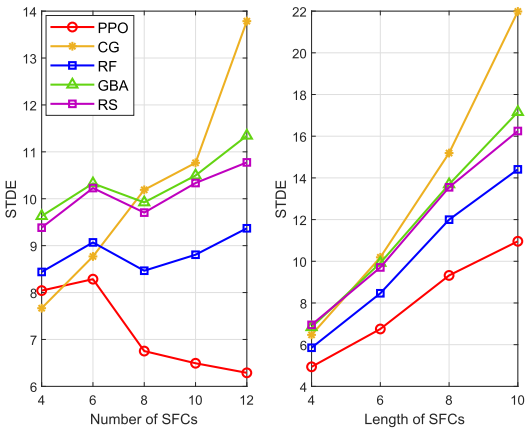


Fig. 9. Performance evaluation of fairness in different settings.

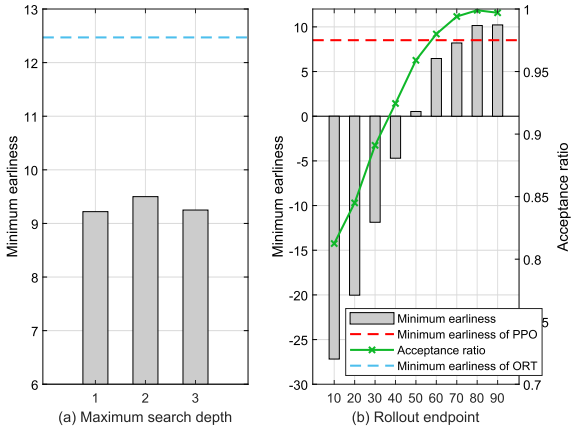


Fig. 10. Average minimum earliness and acceptance ratio of the PPO-MCTS in different settings. $N_u = N_a = 10$, $\mu = 2$ and $|\mathcal{F}_i| = 10, \forall i \in \mathcal{N}_a$.

length of SFCs increases, the STDE of the benchmarks also increase while that of the PPO algorithm increases the least. Meanwhile, the STDE of the PPO algorithm even decreases with the increasing SFCs, which indicates that the PPO algorithm can effectively guarantee the fairness and shows good robustness in different settings.

C. Performance of the PPO-MCTS Algorithm

In the PPO-MCTS, the decisions are expected to deliver a more compact schedule with lookahead steps. However, adding the search depth increases the search space exponentially. We need to know how deep an MCTS is enough to obtain a good result. Moreover, a small number of simulation steps cannot provide an accurate evaluation on the cumulative reward of a leaf node, while a large number of simulation steps increases the computational cost. Therefore, we need to determine the simulation endpoint to achieve a trade-off between an effective evaluation and an efficient search. Since the PPO algorithm have growing performance advantages with increasing problem size, we investigate the performance gap between the our algorithms with the solver for large-scale problems.

Fig. 10(a) shows the average minimum earliness for different maximum search depths. The number of iterations is

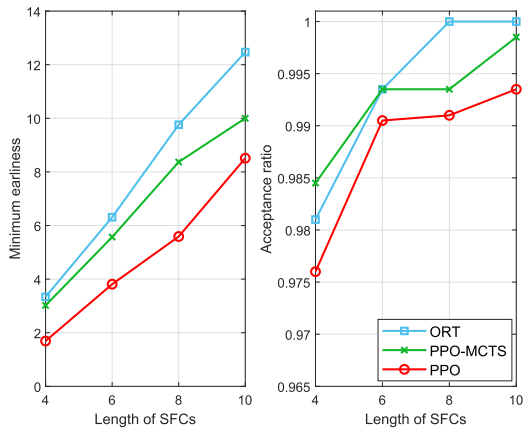


Fig. 11. Average minimum earliness and acceptance ratio as a function of the length of SFCs. $N_u = N_a = 10$ and $\mu = 2$.

$q = 50$ and the simulation endpoint is $h = 80$. We can observe similar performance in each setting and the performance gap with the solver does not change significantly at different depths. The reason is that these search depths are not deep enough to capture the difference between different lookahead steps. However, larger search depths require more iterations to explore the entire search space, which is computationally prohibitive. We set the maximum search depth to 1. Since the simulation results of a deterministic policy are definite for a fixed start state and simulation endpoint, we set the number of iterations to be the same as the size of the action space $|\mathcal{A}_n|$ to further reduce the computational overhead in an MCTS search.

Fig. 10(b) shows the average minimum earliness and acceptance ratio with the increasing simulation endpoint, where the maximum search depth is $d = 1$. The simulation endpoint varies from 10 to 90. As we can see, the minimum earliness and the acceptance ratio increase with the simulation endpoint, because the evaluations of the nodes are approaching the long-term performance. The minimum earliness of the PPO-MCTS is even inferior to the PPO algorithm when the endpoint is close to the root. The reason is that while simulating with the policy network, the cumulative reward over a short period does not consider the influences of an action on future decisions, whose inaccuracy may mislead the decision of MCTS. When the simulation endpoint reaches 80, the performance improvement is not significant. On average, each decision combines the influence of the next four decisions for each SFC. Therefore, we set $4(N_u + N_a)$ as the simulation endpoint for N_u unfinished SFCs and N_a arrived SFCs.

Fig. 11 shows the average minimum earliness and acceptance ratio as a function of the length of SFCs. The solutions of the proposed methods are benchmarked with those obtained by ORT. The PPO-MCTS obtains a higher minimum earliness than the PPO algorithm because the planning procedure is equivalent to taking the constraints between the VNFs of future mapping and scheduling into account to make a decision in current step. Note that the acceptance ratio of PPO-MCTS is higher than that of ORT when the length of SFCs is 4. The reason is that maximizing minimum earliness may reduce the severities of timeout for certain SFCs but defer the completions of the SFCs that are about to exceed the deadline.

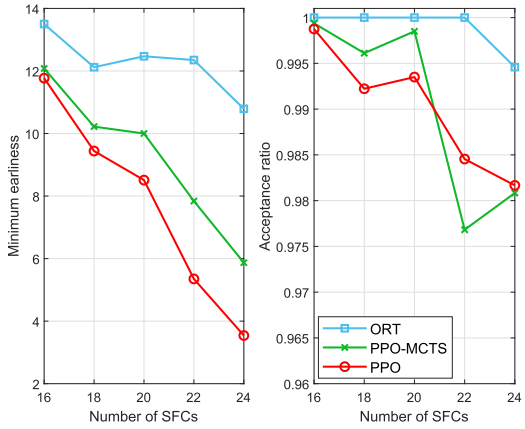


Fig. 12. Average minimum earliness and acceptance ratio as a function of the number of SFCs. $\mu = 2$ and $|\mathcal{F}_i| = 10, \forall i \in \mathcal{N}_a$.

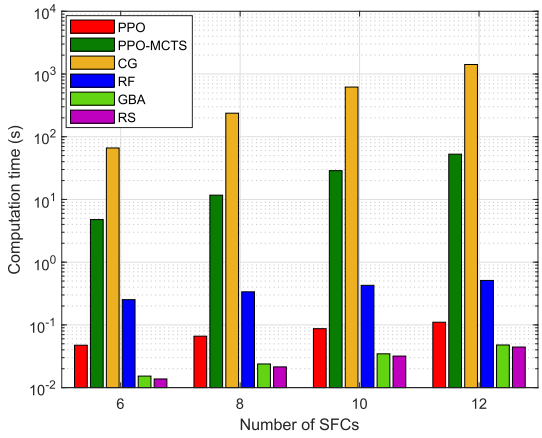


Fig. 13. Average computation time as a function of the number of SFCs. $\mu = 2$ and $|\mathcal{F}_i| = 6, \forall i \in \mathcal{N}_a$.

Fig. 12 shows the average minimum earliness and acceptance ratio as a function of the number of SFCs. The PPO-MCTS obtains a higher performance than the PPO algorithm. However, the acceptance ratio of the PPO-MCTS is even lower than the PPO algorithm when the number of SFCs is equal to 22 and 24. The reason is that the number of VNF types is definite, and as the number of SFCs increases, the probability that SFCs have common VNFs also increases. This increases the competitions over the finite VNF instances and defers the completions of all SFCs, thereby making the completion time more concentrated around the deadline. Maximizing minimum earliness defers their completions and results in more delays.

D. Computation Time

Fig. 13 shows the average computation time of all the considered methods. As can be seen from Fig. 13, the average computation time of the PPO is about 3 orders of magnitude lower than that of the CG, and is 20% of the RF-based method. The reason is that the weights of the policy network are shared across different actions in a problem. The computational complexity is only proportional to the product of the number of VNFs and the size of action space, which is $N_f(N_u + \mu N_a)$. Note that the proposed PPO algorithm can be accelerated by using parallel computation processers such as GPU. As the number of SFCs increases, the running time of proposed PPO algorithm tends to be of the same order of magnitude as the RS algorithm. As for the PPO-MCTS algorithm, the computational overhead is about 2 orders of magnitude higher than that of the PPO algorithm. Thus, the high performance and speedup provided by the PPO algorithm makes it a promising candidate for practical implementation. The PPO-MCTS algorithm can be used to improve fairness performance when the requirements of computation time are not stringent.

VI. Conclusion

In this paper, we have investigated the VNF mapping and scheduling problem with the objective of maximizing the fairness of different services while ensuring the corresponding delay requirements. We characterize the optimization task in a Markov decision process and propose a deep RL method with offline PPO to learn the VNF mapping and scheduling policy. The decisions of mapping and scheduling are incorporated into one action space to answer the need of considering the interplay. The policy is designed to share parameters across all actions, making it scalable to the number of services. Numerical results demonstrate that the proposed algorithm outperforms the traditional random forest algorithm and greedy ones in terms of the fairness and acceptance ratio. We further improve the policy by introducing the MCTS into the decision process. The deterministic characteristics of the VNF mapping and scheduling problem are utilized to provide a lookahead search by simulating with the learned policy. Extensive experiments show that the proposed PPO-MCTS scheme can achieve a fairness improvement.

Acknowledgment

The authors would like to thank the editors and the anonymous reviewers, whose invaluable comments helped improve the presentation of this paper substantially.

References

[1] Z. Kuai and S. Wang, "Fair virtual network function scheduling with deep reinforcement learning," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2021, pp. 1–6.
[2] M. Giordani, M. Polese, M. Mezzavilla, S. Rangan, and M. Zorzi, "Toward 6G networks: Use cases and technologies," *IEEE Commun. Mag.*, vol. 58, no. 3, pp. 55–61, Mar. 2020.
[3] J. G. Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Trans. Netw. Service Manage.*, vol. 13, no. 3, pp. 518–532, Sep. 2016.
[4] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Commun. Mag.*, vol. 53, no. 2, pp. 90–97, Feb. 2015.
[5] R. Mijumbi *et al.*, "Network function virtualization: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 236–262, 1st Quart., 2016.
[6] 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.
[7] L. Gu, J. Hu, D. Zeng, S. Guo, and H. Jin, "Service function chain deployment and network flow scheduling in geo-distributed data centers," *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 4, pp. 2587–2597, Oct. 2020.

[8] S. Ayoubi, S. Sebbah, and C. Assi, "A logic-based benders decomposition approach for the VNF assignment problem," *IEEE Trans. Cloud Comput.*, vol. 7, no. 4, pp. 894–906, Dec. 2019.
[9] J. Li, W. Shi, N. Zhang, and X. Shen, "Delay-aware VNF scheduling: A reinforcement learning approach with variable action set," *IEEE Trans. Cognit. Commun. Netw.*, vol. 7, no. 1, pp. 304–318, Mar. 2021.
[10] 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. (INFOCOM)*, May 2021, pp. 1–10.
[11] Y. Zhang, F. He, T. Sato, and E. Oki, "Network service scheduling with resource sharing and preemption," *IEEE Trans. Netw. Service Manage.*, vol. 17, no. 2, pp. 764–778, Jun. 2020.
[12] L. Qu, C. Assi, and K. Shaban, "Delay-aware scheduling and resource optimization with network function virtualization," *IEEE Trans. Commun.*, vol. 64, no. 9, pp. 3746–3758, Sep. 2016.
[13] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and S. Davy, "Design and evaluation of algorithms for mapping and scheduling of virtual network functions," in *Proc. 1st IEEE Conf. Netw. Softwarization (NetSoft)*, Apr. 2015, pp. 1–9.
[14] T. Gao *et al.*, "Cost-efficient VNF placement and scheduling in public cloud networks," *IEEE Trans. Commun.*, vol. 68, no. 8, pp. 4946–4959, Aug. 2020.
[15] H. A. Alameddine, S. Sebbah, and C. Assi, "On the interplay between network function mapping and scheduling in VNF-based networks: A column generation approach," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 4, pp. 860–874, Dec. 2017.
[16] H. A. Alameddine, M. H. K. Tushar, and C. Assi, "Scheduling of low latency services in softwarized networks," *IEEE Trans. Cloud Comput.*, vol. 9, no. 3, pp. 1220–1235, Jul. 2021.
[17] T. Wang, S. Wang, and Z.-H. Zhou, "Machine learning for 5G and beyond: From model-based to data-driven mobile wireless networks," *China Commun.*, vol. 16, no. 1, pp. 165–175, Jan. 2019.
[18] T. Wang, W. Yu, and S. Wang, "Inter-slice radio resource management via online convex optimization," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2021, pp. 1–6.
[19] T. Wang and S. Wang, "Inter-slice radio resource allocation: An online convex optimization approach," *IEEE Wireless Commun.*, vol. 28, no. 5, pp. 171–177, Oct. 2021.
[20] T. Wang and S. Wang, "Online convex optimization for efficient and robust inter-slice radio resource management," *IEEE Trans. Commun.*, vol. 69, no. 9, pp. 6050–6062, Sep. 2021.
[21] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Math. Oper. Res.*, vol. 1, no. 2, pp. 117–129, May 1976.
[22] V. Sels, N. Gheysen, and M. Vanhoucke, "A comparison of priority rules for the job shop scheduling problem under different flow time-and tardiness-related objective functions," *Int. J. Prod. Res.*, vol. 50, no. 15, pp. 4255–4270, 2012.
[23] F. Pezzella, G. Morganti, and G. Ciaschetti, "A genetic algorithm for the flexible job-shop scheduling problem," *Comput. Oper. Res.*, vol. 35, no. 10, pp. 3202–3212, 2008.
[24] C. Zhang, W. Song, Z. Cao, J. Zhang, P. S. Tan, and X. Chi, "Learning to dispatch for job shop scheduling via deep reinforcement learning," in *Proc. NeurIPS*, 2020, pp. 1–12.
[25] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.
[26] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. ICML*, 2016, pp. 1928–1937.
[27] C. B. Browne *et al.*, "A survey of Monte Carlo tree search methods," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 1–43, Mar. 2012.
[28] C. D. Rosin, "Multi-armed bandits with episode context," *Ann. Math. Artif. Intell.*, vol. 61, no. 3, pp. 203–230, 2011.
[29] D. Silver *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.
[30] S. Jun, S. Lee, and H. Chun, "Learning dispatching rules using random forest in flexible job shop scheduling problems," *Int. J. Prod. Res.*, vol. 57, no. 10, pp. 3290–3310, May 2019.
[31] L. Perron and V. Furnon. *Or-Tools*. Accessed: Mar. 30, 2021. [Online]. Available: https://developers.google.com/optimization/

Zhenran Kuai received the B.S. degree from Nanjing University, Nanjing, China, in 2019, where he is currently pursuing the Ph.D. degree with the School of Electronic Science and Engineering. His current research interests include network function virtualization and machine learning.

Tianyu Wang (Member, IEEE) received the Ph.D. degree from the School of Electronics Engineering and Computer Science, Peking University, Beijing, China, in 2016. He is currently an Assistant Professor with the School of Electronic Science and Engineering, Nanjing University, China. His current research interests focus on network slicing and machine learning in wireless networks.

Shaowei Wang (Senior Member, IEEE) received the Ph.D. degree from Wuhan University, Wuhan, China, in 2006. He joined the School of Electronic Science and Engineering, Nanjing University, Nanjing, China, as a Faculty Member, in 2006, where he is currently a Full Professor. From 2012 to 2013, he was a Visiting Scholar/a Professor with Stanford University, Stanford, CA, USA, and The University of British Columbia, Vancouver, BC, Canada. His research interests include communications and networking, operations research, and machine learning.