Policy Network Assisted Monte Carlo Tree Search for Intelligent Service Function Chain Deployment

Zhihan Fu*, Qilin Fan*, Xu Zhang†, Xiuhua Li*, Sen Wang* and Yueyang Wang*
*School of Big Data and Software Engineering, Chongqing University, Chongqing, China
†College of Engineering, Mathematics & Physical Sciences, University of Exeter, UK
Email: *{fuzhihan,fanqilin,wangsen,yueyangw}@cqu.edu.cn,lixiuhua1988@gmail.com,†X.Zhang6@exeter.ac.uk

***Abstract*—Network function virtualization (NFV) simplifies the configuration and management of security services by migrating the network security functions from dedicated hardware devices to software middle-boxes that run on commodity servers. Under the paradigm of NFV, the service function chain (SFC) consisting of a series of ordered virtual network security functions is becoming a mainstream form to carry network security services. Allocating the underlying physical network resources to the demands of SFCs under given constraints over time is known as the SFC deployment problem. It is a crucial issue for infrastructure providers. However, SFC deployment is facing new challenges in trading off between pursuing the objective of high revenue-to-cost ratio and making decisions in an online manner. In this paper, we investigate the use of reinforcement learning to guide online deployment decisions for SFC requests and propose a Policy network Assisted Monte Carlo Tree search approach named PACT to address the above challenge, aiming to maximize the average revenue-to-cost ratio. PACT combines the strengths of the policy network, which evaluates the placement potential of physical servers and the Monte Carlo Tree Search, which is able to tackle problems with large state spaces. Extensive experimental results demonstrate that our PACT achieves the best performance and superior to other algorithms by up to 30% and 23.8% on average revenue-to-cost ratio and acceptance rate, respectively.**

I. Introduction

With the rapid development of computer network technology and its wide range of application scenarios, network security has attracted more attention. How to ensure the security of the network has become an important topic. Security management is a tedious and manual process involving the implementation, deployment, operation, and maintenance of diverse security functions. Traditionally, security functions are deployed on specialized hardware appliances, which have short life cycles and are usually expensive and customized. In addition, dynamically adding new security function is usually cumbersome and challenging [1].

Network function virtualization (NFV) has emerged as a promising network paradigm which is promoted by the European Telecommunications Standards Institute (ETSI) [2]. By taking advantages of virtualization techniques and high capacity servers. NFV decouples the implementation of network security functions, such as firewall, load balancer, deep packet inspection (DPI) and intrusion prevention system (IPS), from dedicated hardware devices and implements them in the form of software middle-boxes that run on top of commodity servers [3]. Under the paradigm of NFV, network security functions or applications are deployed in the form of virtual network functions (VNFs). For example, network traffic may need to pass through a set of VNFs in a specific order to achieve a security service (e.g. Firewall⇒DPI⇒IPS), which is known as the service function chain (SFC) [4]. Infrastructure providers (InPs) usually need to dynamically create and configure diverse SFCs to observe, filter or monitor the network traffic and address particular security requirements for different application types.

Due to the various advantages of NFV, NFV-based SFC has become a primary form of network service. However, SFC deployment, which allocates resources of underlying physical network to the demands of SFCs under given constraints over time, is a crucial and challenging issue for InPs as it has a direct effect on the network performance and revenue but is proved to be NP-hard [2]. Many authors have proposed exact approaches to obtain the optimal solution for the small instances of the SFC deployment problem [5], [6]. However, these approaches are computationally expensive, which makes them not suitable for large scale network. Therefore, numerous heuristic-based approaches are proposed [7], [8],which try to get the optimal solution within an acceptable time. Nevertheless, these heuristic-based algorithms lack adaptability to various environments. In recent years, reinforcement learning has show super performance in dealing with complex tasks. A significant number of reinforcement learning-based approaches are proposed to solve the SFC deployment [9], [10]. These approaches show great effectiveness, but their effectiveness will decrease when the space of actions is enormous. In summary, there is an urgent requirement to design a new intelligent approach to balance pursuing the optimal deployment and making decisions in an online manner.

In this paper, we investigate the use of RL to guide online deployment decisions for SFC requests. To this end, we propose a **P**olicy network **A**ssisted Monte **C**arlo **T**ree search approach, named PACT, to address the above challenge. PACT combines the strengths of the policy network which evaluates the potentials of physical servers and the Monte Carlo Tree Search (MCTS) which is able to tackle problems with large state spaces. The effectiveness of PACT is evaluated by extensive simulations. The main contributions of this paper are summarized as follows:

- We investigate the online SFC deployment problem in the NFV-enabled network. The problem is proven to be an NP-hard problem and formulated as a Markov decision

process (MDP) with carefully-designed states, actions, rewards and transitions.

- We design a RL based approach (PACT) which approximates optimal decisions with a search tree using prioritized samples learned by policy network in the decision space. It could seek a trade-off between optimizing efficiency and precision.
- We conduct comprehensive simulations to evaluate the efficiency of the proposed algorithm. The results demonstrate that our algorithm achieves the best performance, and superior to other algorithms by up to 30% and 23.8% on average revenue-to-cost ratio and acceptance rate, respectively.

The rest of the paper is organized as follows. We describe the related work in Section II. We present the system model and problem formulation of SFC deployment in Section III. We define the problem as a MDP, and give our PACT approach in Section IV. The simulation results are shown in Section V. Finally, we summarize the paper in Section VI.

II. Related Work

The SFC deployment problem has been extensively studied in recent years. Comprehensive surveys are already given in [2] and [11]. In this section, we summarize the related works and categorize them into the following three groups.

A. Exact approach

In recent years, a significant number of papers modelling the SFC deployment problem as a mathematical model in the case of one or multiple objectives and utilize the exact approaches to obtain the optimal solution for the small instances of the problem. We can divide these works into different categories according to their mathematical model. The commonly used mathematical models are integer linear program (ILP), mixed-integer linear program (MILP), and binary integer linear program (BILP). Tomassilli *et. al.* [5] formulated the SFC deployment problem as an ILP problem with the objective to minimize the amount of used physical servers, and proposed a randomized rounding method to solve it. Jang *et. al.* [6] modeled the SFC deployment problem as a MILP problem with the objective of minimizing the energy cost and utilized a linear relaxation and rounding method to find the optimal solution. Mijumbi *et. al.* [12] formulated the SFC deployment problem as a BILP problem with the objective to minimize the server setup cost, and proposed a greedy approximation method to solve it. However, due to the dependence of prior knowledge of SFCs and the high computational complexity, these offline approaches are difficult to apply to online placement scenarios.

B. Heuristic based approach

A variety of heuristic-based approaches have been proposed in the current works. Heuristic-based algorithms usually leverage some artificial rules to reduce the search space, which is generally effective and could be implemented in actual scenarios. Kuo *et. al.* [7] proposed a dynamic programming based heuristic algorithm to solve the SFC deployment problem, which follows the guidance of the link and server usage to maximize the resource utilization in the offline scene. Khebbache *et. al.* [8] proposed a multi-stage graph construction method to solve the SFC deployment with the objective of maximizing the acceptance rate in the online situation. Liu *et. al.* [13] proposed a novel heuristic dynamic programming based cost optimization algorithm to solve the SFC deployment to minimize the server running cost and communication cost. Heuristic based algorithms might perform well in the network where the environment is not dynamic. When considering the online scenario, heuristic solutions might fall into the local optimum.

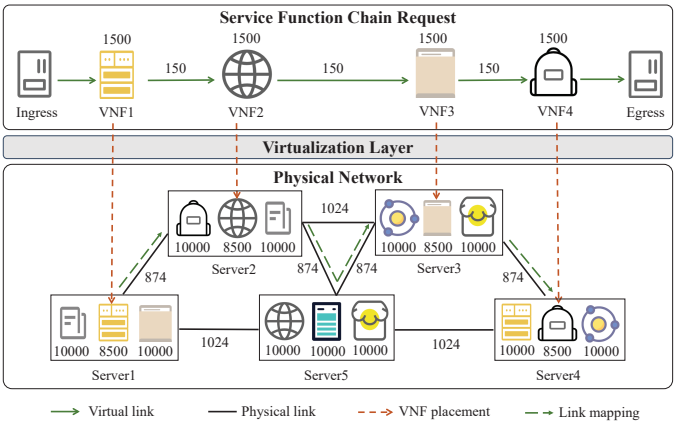

Fig. 1. An example of SFC deployment.

C. Reinforcement learning (RL) based approach

Nowadays, RL based approaches have shown great potential to deal with combinatorial optimization problems. Mijumbi *et. al.* [9] proposed a distributed Q-Learning based algorithm for SFC deployment to maximize the acceptance rate. Khezri *et. al.* [10] proposed a deep Q-network (DQN) based approach for SFC deployment with the objective to minimize the deployment cost. Xiao *et. al.* [14] proposed a deep reinforcement learning approach for SFC deployment with the objective to minimize the operation cost and maximize the acceptance rate jointly. Pei *et. al.* [15] proposed a double deep Q-network (DDQN) based approach for SFC deployment with the objective to minimize the end-to-end delay. However, these approaches have to deal with huge search space, which makes them not suitable for online decision.

III. System Model and Problem Formulation

In this section, we conduct a detailed analysis of the SFC deployment in the NFV-enabled network. The key notations used in this paper are summarized in Table I.

A. Physical Network Model

We consider an undirected graph $G = (V, E)$ to represent the NFV-enabled substrate/physical network, where V denotes the set of physical servers and E denotes the set of physical links. For any link $e \in E$, let B_e represent its remaining bandwidth. Each server $v \in V$ hosts a certain amount virtual machines (VMs), which is denoted as N_v, and the set M

TABLE I. SUMMARY OF KEY NOTATIONS

Notation	Description
Network	G(V, E)
V	The set of physical servers, with $v \in V$
E	The set of physical links, with $e \in E$
B_e	The remaining bandwidth of physical link e
N_v	The number of VMs on each server
M	The set of VMs hosted on different servers, with $m \in M$
C_m	The remaining computing capability of VM m
F	The set of all VNF types supported by the physical network
SFC	s^k
μ	The arrival rate of SFC request
F^{s^k}	The VNF set requested by SFC s^k
$c_{f_t^k}$	The computing demand of the t-th VNF of k-th SFC request
b^k	The bandwidth demand of SFC request k
a_k	The arrival time of SFC request k
l_k	The lifetime of SFC request k
o^k	The ingress vertex of the SFC request k
d^k	The egress vertex of the SFC request k with $f_t^k \in F^{s^k}$
p^k	The selected mapping path of SFC request k
MDP	$\mathcal{M}^k$
ϕ_t^k	The state of the physical network and SFC request k at time t
A_t^k	The action set of deploying SFC request k at time t
R_t	The immediate reward after selecting a server v_t for placement f_t^k at time t

represent all VMs hosted on different servers. For each VM $m \in M$, let C_m denote its remaining computing capability. We assume that the physical network can support a set of VNFs F and each VM $m \in M$ can only run one type of VNF $f \in F$. However, different VMs might run the same type of VNF. Besides, we assume that the bandwidth between VMs on the same server v is sufficient to support the maximum bandwidth, allowing different VNFs of the same SFC to be deployed on the same server. An example of a physical network is illustrated in the bottom of Fig. 1. The physical network contains five servers and seven links. Each server consists of three VMs. The numbers around the VMs and links are the available resources for them.

B. SFC Request Model

In this paper, we use μ to denote the arrival rate of SFC requests, which follows a Poisson distribution. Let $s^k = \{f_1^k, \ldots, f_t^k, \ldots, f_{T^k}^k\}$ represent the k-th SFC request. The set of VNFs for SFC request s^k is denoted as F^{s^k}=$\{ f_t^k | t \in [1, T^k]\}$, where f_t^k is the t-th VNF of k-th SFC request. For each VNF f_t^k, let $c_{f_t^k}$ indicate the computing demand of it. As F represents the set of all VNF types supported by the network, we assumed that any VNF $f_t^k \in F$ is used at most once in each SFC request to ensure that no duplicate type of VNF occurs. We assume that each SFC request s^k has a bandwidth demand b_k, o^k and d^k are denoted as the ingress vertex and egress vertex of SFC request s^k, respectively. Moreover, let a^k and l^k be the arrival time and lifetime of SFC request s^k, respectively. An example of a SFC request is illustrated in the upper of Fig. 1. The SFC request contains four different VNFs and connected by three virtual links. The numbers around the VNFs and virtual links are the requested resources.

C. Problem Formulation

The main objective of a SFC deployment is to effectively allocate resources of the underlying physical network to the demands of SFCs under given constraints over time.

1) Objective Function: To maximize the InP's profit, which is defined as the ratio between the generated revenue and the placement cost, like previous work [16], we aim to maximize the revenue-to-cost ratio, which is defined as:

$$\mathcal{F}(s^k) = \begin{cases} \dfrac{R(s^k)}{C(s^k)}, & \text{if } s^k \text{ is accepted,} \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Revenue: The revenue of accepting a SFC request s^k can be defined as:

$$R(s^k) = \left[\eta \cdot \sum_{f_t^k \in s^k} c_{f_t^k} + \beta \cdot (|s^k| - 1) \cdot b_k \right] \cdot l^k, \tag{2}$$

where η and β are the unit prices charged for the computing resource and bandwidth resource, which are determined by the InPs [17].

Cost: After deploying a SFC request s^k, the InP allocates physical resources for the SFC and incurs a cost, which is defined as:

$$C(s^k) = \left[\sum_{f_t^k \in s^k} c_{f_t^k} + |p^k| \cdot b_k \right] \cdot l^k, \tag{3}$$

where $|p^k|$ is the length of the mapping path. It is worth noting that if two consecutive VNFs of a SFC request are placed in the same server, the bandwidth cost will be ignored.

2) VNF Placement Constraint: Each VM $m \in M$ can only support one type of VNF $f \in F$. However, it can be shared by different requests. That is, if several VNFs that belong to different SFC requests are allocated to the same VM, those VNFs must have the same type of VNF. In other words, if $m(f_t^k) = m(f_{t'}^{k'})$, then $f_t^k = f_{t'}^{k'}, \forall k, k', t, t'$. Therefore, a VM is eligible to host a VNF f_t^k if:

$$f(m) = f_t^k, \tag{4}$$

$$c_{f_t^k} \leq C_m, \ \forall m \in M, \tag{5}$$

where $f(m)$ is the type of VNF that VM m runs.

3) Link Capacity Constraint: For two adjacent VNFs in the SFC request, we need to find the path between the two

mapped servers which can satisfy the bandwidth requirement of the SFC request. For each SFC request s^k, the selected mapping path p^k should carry the bandwidth requirement b_k and traverse through VNFs following the order specified in the request. Since s^k might be deployed on a path with loops we need consider the total bandwidth consumed by the path p^k. We have:

the path mapped in the physical network

$$\delta_e(p^k) \cdot b_k \leq B_e, \forall e \in E, \tag{6}$$

where $\delta_e(p^k)$ represents the counts that path p^k passes through link e.

Theorem 1: The SFC deployment problem is NP-hard.

Proof 1: The multiple knapsack (MK) problem is known to be an NP-hard problem [18]. To demonstrate the SFC deployment is NP-hard, we transform the SFC deployment problem into an MK problem. Given a set of I items denoted as set $\mathcal{I} = \{1, 2, ..., I\}$ with profit p_i and weight w_i for each item i, and a set of J knapsacks denoted as set $\mathcal{J} = \{1, 2, ..., J\}$ with capacity c_j for each knapsack $j \in \mathcal{J}$. The object is to maximize the total profit $\sum_{j=1}^{J}\sum_{i=1}^{I} p_i x_j^i$ subject to the knapsack capacity constraint $\sum_{i=1}^{I} w_i x_j^i \leq c_j, \forall j \in \mathcal{J}$, where x_j^i indicates whether item i is assigned to the knapsack j.

Then we transform the SFC deployment problem into an MK problem. SFC deployment mainly includes two stages, namely, VNF placement stage and link mapping stage. First, we only consider the VNF placement stage. If the VNF placement stage is NP-hard, the SFC deployment problem can be proved to be NP-hard. For each item set $\mathcal{I}$, we use an SFC request s^k with I VNFs to represent, thus the computing demand $c_{f_t^k}$ is equal to w_i. For each knapsack j, we use the physical server v, which host only one VM with computing capacity C_m to represent, thus C_m is equal to c_j. As the objective of the VNF placement is maximizing the revenue-to-cost ratio (the total profit of the MK). The VNF placement problem is transformed into an MK problem. Therefore, the SFC deployment problem is an NP-hard problem.

IV. Policy network Assisted monte Carlo Tree search approach

In this section, we first model the SFC deployment problem as a MDP. Then we introduce our proposed PACT algorithm, which in NFV-enabled networks.

A. MDP Model for SFC Deployment

Nowadays, reinforcement learning approaches have shown their great potentials to deal with combinatorial optimization problems. In standard reinforcement learning model, the learning agent interacts with the environment and improves its performance based on its own experience. In this paper, we model the online SFC deployment as a sequential decision-making problem, which can be formulated as a MDP. The agent continuously solves the VNF placement and virtual link mapping for each SFC request s^k. The objective of the agent is to seek a deployment strategy that can maximize the long-term average revenue-to-cost ratio.

At a given time point, the SFC solver agent receives a SFC request s^k. We present the SFC deployment of s^k as a finite-horizon MDP $\mathcal{M}^k$ as follows:

- **States:** The state of $\mathcal{M}^k$ at a time step t is defined as:

$$\phi_t^k = (F_t^k = F_{t-1}^k \backslash \{f_{t-1}^k\}, V_t(f_t^k)), \tag{7}$$

  where F_t^k is the ordered set of VNFs that are yet to be placed and f_{t-1}^k is the VNF that have been placed in the previous time step. $V_t(f_t^k)$ is the set of all candidate servers that are available for placement f_t^k at state ϕ_t^k.
- **Actions:** An action a selects a physical server v_t from the candidate server set $V_t(f_t^k)$ to place the VNF f_t^k. The set of actions is defined as:

$$A_t^k = \{\varpi\} \cup \{(f_t^k, v_t) : \forall v_t \in V_t(f_t^k)\}, \tag{8}$$

  where ϖ represents the action that forces the transition to the terminal state when $V_t(f_t^k) = \varnothing$.
- **Transitions:** There are $|A_t^k|$ possible next states based on the selection of physical server for the placement of VNF f_t^k at time step t.
- **Rewards:** After selecting a server v_t for placement f_t^k at time step t, the agent receives a reward R_t according to Eq. (1). As partially placement of SFC does not necessarily lead to complete deployment successfully, the immediate reward R_t for all time step $t \leq T^k$ are set zero in this paper.
- **Policy:** The objective of the agent is to seek a policy π^* that can maximize the overall rewards from initial state.

B. PACT Algorithm

The design of our proposed PACT is shown in Fig. 2. It mainly including two phases, namely, offline training and online decision-making.

1) Offline Training: In order to make full use of historical data of SFC requests, we train a policy network (PN) [19] to evaluate the placement potential of each physical server, which consists of an input layer, a convolutional layer, a softmax layer and a filter layer. The pseudocode of PACT offline training process is shown in **Algorithm 1**.

PN takes the feature matrix $X \in \mathbb{R}^{N*H}$ as the input, where $N = |V|$ is the number of physical servers and the i-th row of X refers to the feature vector with H-dimensions of physical server i. Here, we extract remaining computing capability, number of adjacent links, and the sum of remaining bandwidth of adjacent links as features of each server, and normalize them into $[0, 1]$. Then PN passes the X to the convolutional layer with a convolution kernel and output a vector representing the deployment potential of each server:

$$v_i = w \cdot x_i + b, \tag{9}$$

where v_i is the i-th output of the convolutional layer, w is the convolution kernel weight vector, and b is the bias. The vector is subsequently passed to the softmax layer which transforms the values of neurons to a probability distribution. Finally, PN uses the filter layer to filter out servers that do not meet the requirements and output the best server.

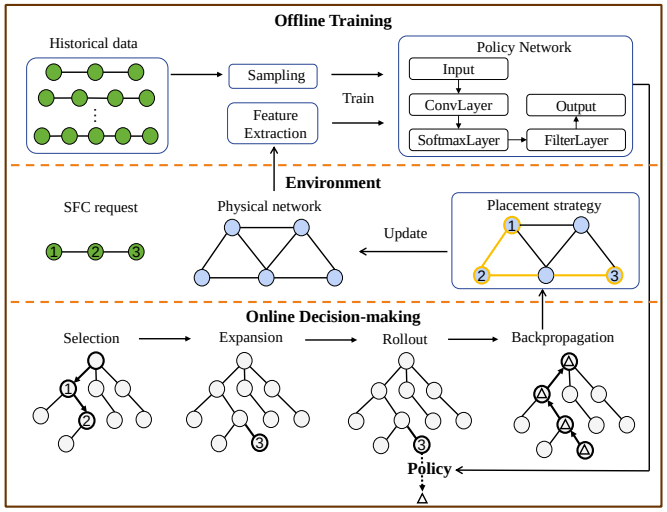

Fig. 2. The process of SFC deployment based on PACT.

To better train the neural network, we assume that every decision that the model makes to be correct and assign a label to each output. That is, if we choose the i-th server, the manual label would be a zero-filled vector y except the i-th position which is one. We try to minimize the cross-entropy loss. After placing the t-th VNF of the historical SFC request $s^{k'}$, we compute the gradient g_t and stack it. If the request fails to be deployed, the stacked gradients will be discarded. Otherwise, we multiply the stacked gradients $\sum_{t=1}^{T^{k'}} g_t$ with revenue-to-cost ratio r and learning rate α to achieve the final gradients $g^f = \alpha \cdot r \cdot \sum_{t=1}^{T^{k'}} g_t$. Finally, we update the policy network parameters when the batch size is reached.

2) Online Decision-making: Integrated with multi-armed bandit algorithm which is able to seek a balance between exploration and exploitation, MCTS has shown its great potential on solving MDP with large state spaces [17]. MCTS builds a sparse search tree and uses Monte Carlo simulation to select actions to deepen the tree in the most promising direction, which guarantees asymptotic convergence to the optimum. Therefore, PACT utilizes MCTS algorithm for online decision-making, shown in **Algorithm 2**.

We consider a search tree for solving MDP $\mathcal{M}^k$, whose depth is represented by the horizon T^k. MCTS treats the current state as a root to progressively establish the search tree and explores more promising area of the search tree. The tree is initialized with the root node, which represents the initial state of $\mathcal{M}^k$ (shown as Lines 1-2). The nodes and edges in the search tree represent the states and actions, respectively. A policy π is defined as the path from the root to a leaf node. MCTS continues to carry out the four steps (i.e., selection, expansion, rollout and backpropagation) until the computational budget θ is achieved. The computational budget θ represents the number of evaluated action samples per selection cycle. After repeating the four steps θ times, the child of the root with the highest average value is selected as the optimal action and the MDP enters its next state. The

Algorithm 1 PACT's offline training.

Input: Physical network $G = (V, E)$, Learning rate r, Epoch $totalEpoch$, Historical data $historicalDate$;

1: Build and initialize the policy network;
2: **for** $episode \leftarrow 1$, $totalEpoch$ **do**
3: $count = 1$;
4: $finalGradients[\,] \leftarrow \varnothing$;
5: **for** $request \in historicalDate$ **do**
6: Sampling SFC request $s^{k'}$;
7: $nodeMap[\,] \leftarrow \varnothing$;
8: **for** VNF $\in s^{k'}$ **do**
9: input feature matrix X of the substrate network to the policy network model;
10: Choose a server v randomly with the probability ϵ , otherwise choose the server v with the highest probability;
11: $nodeMap.add(v)$;
12: compute the gradient g_t and stack it;
13: **end for**
14: **if** $nodeMap.size == |s^{k'}|$ **then**
15: Use Dijkstra algorithm to find the $linkMap$ based on the $nodeMap$;
16: **if** $linkMap$ exist **then**
17: Calculate $reward\ r$ according to (6);
18: $g^f = \alpha \cdot r \cdot \sum_{t=1}^{T^{k'}} g_t$;
19: $finalGradients.add(g^f)$;
20: depoly the SFC request $s^{k'}$ and update G;
21: **end if**
22: **else**
23: clear the stacked gradients g_t ;
24: **end if**
25: $count \leftarrow count + 1$;
26: **if** $count == batchSize$ **then**
27: Using $finalGradients$ update policy network;
28: $count = 1$;
29: $finalGradients[\,] \leftarrow \varnothing$;
30: **end if**
31: **end for**
32: $episode \leftarrow episode + 1$;
33: **end for**

selected child is then chosen to be the new root of the search tree. The four steps are described in details as follows:

- **Selection:** During the selection stage, the algorithm traverses down the tree from the root until a non-terminal and not yet completely expanded leaf node is reached. A tree node is considered completely expanded only when it has one child node for each action (shown as Lines 6-10). The selection strategy should seek a trade-off between the exploitation of states with high value estimates and the exploration of states with uncertain value estimates. Diverse strategies have been proposed in [20], [21]. In this paper, we employs upper confidence bounds for trees (UCT) [20], which is one of the most commonly used selection strategies to select a child κ from:

$$\kappa \in \arg\max_{i \in \mathcal{I}} \left(\frac{n_i}{\sigma_i} + D\sqrt{\frac{\ln \sigma_u}{\sigma_i}} \right), \tag{10}$$

where u denotes the current node of the search tree, $\mathcal{I}$ denotes its children set. n_i denotes the value of node i, and σ_i denotes the visit times of node i. D is the

Algorithm 2 PACT′s online decision-making.

Input: Physical network $G = (V, E)$, SFC request s^k, Computational budget θ;
Output: Placement strategy: $result, nodeMap, linkMap$;
1: $\phi_1 \leftarrow (s^k, G(V, E))$;
2: Create $root(n = 0, \sigma = 0, state = \phi_1)$;
3: $nodeMap[\,] \leftarrow \varnothing$;
4: **while** $root.state$ is non-terminate **do**
5: **while** $\theta > 0$ **do**
6: —*Step* 1. [Selection]
7: κ = $\arg\max_{i \in \mathcal{I}} \left(\frac{n_i}{\sigma_i} + D\sqrt{\frac{\ln \sigma_u}{\sigma_i}} \right)$;
8: **if** $\kappa == \varpi$ **then**
9: **return** $reject$, $\varnothing, \varnothing$;
10: **end if**
11: —*Step* 2. [Expansion]
12: Add child node κ to the search tree;
13: —*Step* 3. [Rollout]
14: **while** $\kappa.state$ is non-terminate **do**
15: Use the parameters of trained policy network to choose action;
16: $\kappa.state = \phi_{next}$;
17: **end while**
18: Calculate $reward$ according to Eq. (1);
19: —*Step* 4. [Backpropagation]
20: **while** $\kappa.parent$ exist **do**
21: $\kappa.n \leftarrow \kappa.n + reward$;
22: $\kappa.\sigma \leftarrow \kappa.\sigma + 1$;
23: $\kappa \leftarrow \kappa.parent$;
24: **end while**
25: $\theta \leftarrow \theta - 1$;
26: $root.\sigma \leftarrow root.\sigma + 1$;
27: $root.n \leftarrow root.n + reward$;
28: **end while**
29: $snI \leftarrow \arg\max_i (\frac{root.child[i].n}{root.child[i].\sigma})$;
30: $root \leftarrow root.child[snI]$;
31: $nodeMap.add(snI)$;
32: **end while**
33: **if** $nodeMap.size == |s^k|$ **then**
34: Use Dijkstra algorithm to find the $linkMap$ based on the $nodeMap$;
35: **if** $linkMap$ exist **then**
36: **return** $accept$, $nodeMap$, $linkMap$;
37: **end if**
38: **end if**
39: **return** $reject, \varnothing, \varnothing$;

exploration constant.

- **Expansion:** When reaching a non-terminate leaf node, one or more of its children can be added to the tree. In this paper, we always add the child node chosen in the current iteration (shown as Lines 11-12). The added node corresponds to the next state.
- **Rollout:** A rollout strategy is used to play the simulated placement until the terminate state is reached. Uniformly randomly selected actions are able to achieve the convergence of MCTS to the optimum given enough time. However, rollout strategy which utilizes the specific domain knowledge can accelerate the convergence speed [22]. Therefore, we use the parameters of the trained policy network to measure the potential of each physical server $v_t \in V_t(f_t^k)$ for the placement of VNF f_t^k, and select the physical server with the highest probability instead of using the random strategy (shown as Lines 13-18).
- **Backpropagation:** When reaching a terminate state in the rollout stage, the reward is calculated and backpropagated to the root node (shown as Lines 19-29).

If each VNF is successfully placed, the process will reach the terminal state when $t = T^k + 1$, and then the agent solves link mapping by using the shortest path algorithm. If link mapping is also successful, the SFC is accepted for the deployment and the agent receives the reward $R_{T^k+1} = R(s^k)/C(s^k)$. Otherwise, the SFC is rejected and $R_{T^k+1} = 0$. If reaching the terminal state when $t \leq T^k$, it means that there is no suitable servers for the placement of VNF f_t^k. This is also implied that the SFC request will be rejected (shown as Lines 30-36).

TABLE II. PARAMETER VALUES

Name	Value	Description
η	1	The unit price of computing resource
β	1	The unit price of bandwidth resource
α	0.005	The learning rate of policy network
D	0.5	The exploration constant
N_v	4	The number of VMs on each server
l^k	100	The SFC request average lifetime
b_k	[150,190]	The SFC request bandwidth demand
M'	100	The batch size
θ	10	The computational budget of PACT

V. PERFORMANCE EVALUATION

In this section, we first describe the experimental settings and then compare the performance of PACT with other algorithms based on the average revenue-to-cost ratio and acceptance ratio. Through performance evaluation results, PACT has been proven to solve the online SFC deployment problem effectively.

A. Experimental Setup

Our experiment follows settings similar to [7]. We use the GT-ITM to generate the NFV-enabled physical network G, which has 60 nodes and 150 links. We set the initial link bandwidth for each link $e \in E$ to 1 Gbps. We generate a set of VNFs F, which contains 30 distinct VNFs. For each VM $m \in M$, its initial computing capability C_m is set to 10000 and $f(m)$ is randomly selected from F.

Each SFC request s^k is constructed by randomly selected 2 to 8 different VNFs from F. We assume the computing demand $c_{f_t^k}$ for each VNF $f_t^k \in s^k$ is associated with the bandwidth demand b_k. Therefore, for each f_t^k , we set $c_{f_t^k} = 10b_k$ and $c_{f_t^k} = 10b_k \ln(b_k)$ with probabilities 90% and 10%, respectively. The arrival rate of the SFC follows the Poisson distribution with an average arrival rate of one per second. When the request leaves, resources are released.

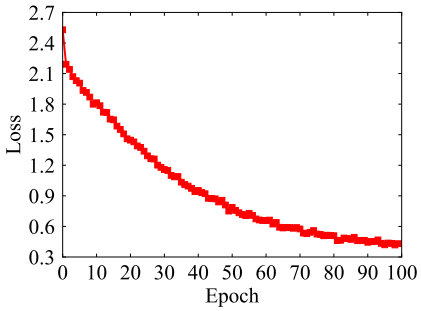

Fig. 3. Loss on the training set.

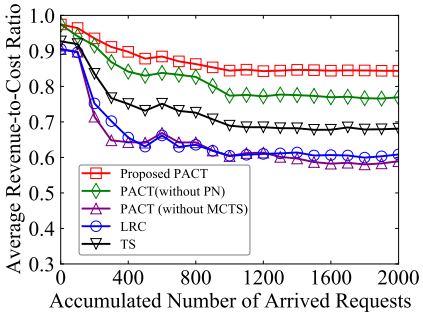

Fig. 4. Performance comparison of the average revenue-to-cost ratio.

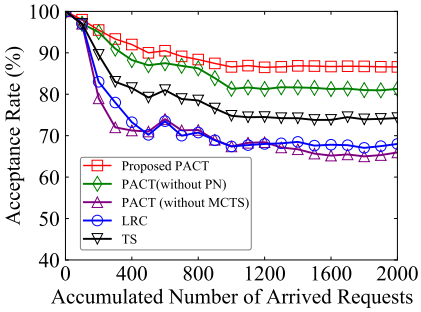

Fig. 5. Performance comparison of the acceptance rate.

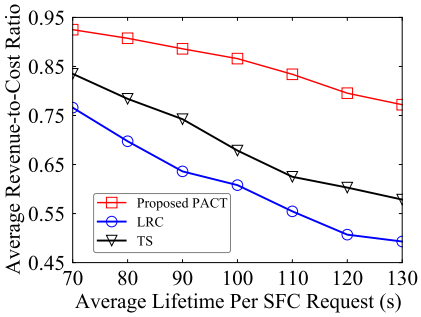

Fig. 6. Average revenue-to-cost ratio with different traffic load.

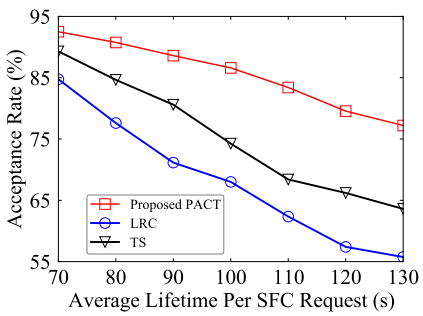

Fig. 7. Acceptance rate with different traffic load.

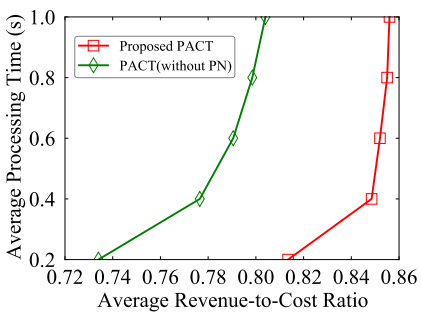

Fig. 8. Performance comparison of the processing time.

We employ TensorFlow to build our policy network and generate 2000 SFC requests that follow the above distribution as the training set to train the model. We use the stochastic gradient descent (SGD) optimizer to update all parameters in policy network and train our network for 100 epochs. All the simulation experiments are executed on a computer with 2.90 GHz Intel Core i7-10700F CPU and 16GB RAM. Table II gives the values of some key parameters.

To evaluate the effectiveness of the proposed PACT, we compare it with the following algorithms:

- **Local Resource Capacity (LRC):** It is a node ranking algorithm proposed in [16], which using LRC metric to map VNFs onto servers in a load-balanced manner.
- **Tarbu Search (TS):** It is a meta-heuristic search method proposed in [23], which using tarbu search to find the least cost SFC deployment strategy.

Both of them utilizes the shortest path algorithm to conduct link mapping.

B. Simulation Results

1) The variation of loss in the training process: We employ decaying ϵ-greedy strategy to train the model. It starts with a high ϵ to conduct better exploration and decreases the ϵ over time to converge smoothly towards the optimal policy. Fig. 3 shows the variation of loss in the training process. It can be observed that the loss decreases through the training stage and reaches stability gradually.

2) Comparison of the average revenue-to-cost ratio: Fig. 4 shows the average revenue-to-cost ratio under different accumulated number of arrived SFC requests. We can see that our proposed PACT algorithm achieves the highest average revenue-to-cost ratio of 0.84, which outperforms PACT (without MCTS), LRC, TS and PACT (without PN) algorithms by up to 30%, 28%, 20% and 9% respectively. This is because by combining the strengths of MCTS and PN, our PACT can better evaluate each placement strategy and make decisions more intelligently.

3) Comparison of the acceptance rate: We use the ratio of the number of accepted SFCs $\mathcal{N}(\hat{s}^k)$ to the number of the arrived SFCs $\mathcal{N}(s^k)$ as the acceptance rate, which is given by:

$$p^a = \frac{\mathcal{N}(\hat{s}^k)}{\mathcal{N}(s^k)}. \tag{11}$$

The acceptance rate under different accumulated number of arrived SFC requests is illustrated in Fig. 5. We can observe that at the beginning, the acceptance ratios of all the four algorithms decrease because the resources of the physical network are gradually occupied as SFC requests arrive. Finally our proposed PACT algorithm achieves the highest acceptance rate of 86.6%, which outperforms PACT (without MCTS), LRC and TS, PACT (without PN) algorithms by up to 23.8%, 21.4%, and 14.2% and 6%, respectively. It indicates that our PACT algorithm can makes better utilization of the substrate resources and reserves more resources for future SFC requests.

4) Impact of the traffic load: In order to estimate the impact of traffic load to the performances of algorithms, we range the average lifetime of SFC requests from 70s to 130s. Fig. 6 and Fig. 7 show that the proposed PACT can achieve better performance compared to TS and LRC algorithms in all cases. When the traffic load is high (i.e., average lifetime is 130s), our proposed PACT still has the highest average revenue-to-cost ratio of 0.76 and acceptance rate of 77%. Even in the case of high workload, the acceptance rate and the average revenue-to-cost ratio of PACT are comparable, indicating that our PACT algorithm has strong robustness.

5) Performance comparison of the processing time: Fig. 8 shows how the average processing time per SFC request varies under the different average revenue-to-cost ratio. It illustrates that the processing time of our proposed PACT is nearly five times faster than the PACT without the assistance of PN when the average revenue-to-cost is 0.813. It demonstrates that the offline training phase could greatly accelerate the processing time for online decision-making.

VI. Conclusion

In this paper, we have proposed an intelligent approach based on reinforcement learning, named PACT for online SFC deployment, to deal with the challenge of trading off between pursuing the objective of high revenue-to-cost ratio and making decisions in an online manner. PACT can approximate optimal decisions with a search tree using prioritized samples learned by policy network in the decision space. Experimental results have shown that our proposed PACT outperforms the other existing algorithms in terms of the average revenue-to-cost ratio and acceptance rate, and can make placement decisions within acceptable time.

Acknowledgements

This work is supported in part by the Major Special Program for Technical Innovation & Application Development of Chongqing Science & Technology Commission (No. CSTC 2019jscxzdztzxX0031), the National NSFC (No. 61902044, 62072060, 61902178, 62002035), the Chongqing Research Program of Basic Research and Frontier Technology (No. cstc2019jcyj-msxmX0589, cstc2018jcyjAX0340), the Natural Science Foundation of Jiangsu (No. BK20190295), the Leading Technology of Jiangsu Basic Research Plan (No. BK20192003), and the European Union's Horizon 2020 Research and Innovation Programme under the Marie Skłodowska-Curie Grant Agreement No. 898588.

References

[1] X. Zhang, H. Huang, H. Yin, D. Wu, G. Min, and Z. Ma, "Resource Provisioning in the Edge for IoT Applications with Multi-Level Services." *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4262–4271, June 2019.

[2] J. Gil-Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, 2016.

[3] Z. Li, Y. Jiang, Y. Gao, L. Sang, and D. Yang, "On Buffer-Constrained Throughput of a Wireless-Powered Communication System," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 2, pp. 283–297, Feb. 2019.

[4] H. Wang, Y. Wu, G. Min, and W. Miao, "A graph neural network-based digital twin for network slicing management," *IEEE Transactions on Industrial Informatics*, doi:10.1109/TII.2020.3047843.

[5] A. Tomassilli, F. Giroire, N. Huin, and S. Pérennes, "Provably efficient algorithms for placement of service function chains with ordering constraints," in *IEEE Conference on Computer Communications (INFOCOM), Honolulu, HI, USA*, Apr. 2018, pp. 774–782.

[6] I. Jang, D. Suh, S. Pack, and G. Dán, "Joint optimization of service function placement and flow distribution for service function chaining," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2532–2541, 2017.

[7] T. Kuo, B. Liou, and K. C. Lin, "Deploying chains of virtual network functions: On the relation between link and server usage," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1562–1576, 2018.

[8] S. Khebbache, M. Hadji, and D. Zeghlache, "Scalable and cost-efficient algorithms for VNF chaining and placement problem," in *20th Conference on Innovations in Clouds, Internet and Networks (ICIN), Paris, France*, Mar. 2017, pp. 92–99.

[9] R. Mijumbi, J. Gorricho, J. Serrat, M. Claeys, F. D. Turck, and S. Latré, "Design and evaluation of learning algorithms for dynamic resource management in virtual networks," in *IEEE Network Operations and Management Symposium (NOMS), Krakow, Poland*, May 2014, pp. 1–9.

[10] H. R. Khezri, P. A. Moghadam, M. Karimzadeh-Farshbafan, V. Shah-Mansouri, H. Kebriaei, and D. Niyato, "Deep reinforcement learning for dynamic reliability aware nfv-based service provisioning," in *IEEE Global Communications Conference (GLOBECOM), Waikoloa, HI, USA*, Dec. 2019, pp. 1–6.

[11] S. Yang, F. Li, S. Trajanovski, R. Yahyapour, and X. Fu, "Recent advances of resource allocation in network function virtualization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 295–314, 2021.

[12] R. Mijumbi, J. Serrat, J. Gorricho, J. Rubio-Loyola, and S. Davy, "Server placement and assignment in virtualized radio access networks," in *11th International Conference on Network and Service Management (CNSM), Barcelona, Spain*, Nov. 2015, pp. 398–401.

[13] Y. Liu, J. Pei, P. Hong, and D. Li, "Cost-efficient virtual network function placement and traffic steering," in *IEEE International Conference on Communications, ICC, Shanghai, China*, May. 2019, pp. 1–6.

[14] Y. Xiao, Q. Zhang, F. Liu, J. Wang, M. Zhao, Z. Zhang, and J. Zhang, "NFVdeep: adaptive online service function chain deployment with deep reinforcement learning," in *Proceedings of the International Symposium on Quality of Service (IWQoS), Phoenix, AZ, USA*, Jun. 2019, pp. 1–10.

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

[16] L. Gong, Y. Wen, Z. Zhu, and T. Lee, "Toward profit-seeking virtual network embedding algorithm via global resource capacity," in *IEEE Conference on Computer Communications (INFOCOM), Toronto, Canada*, Apr. 2014, pp. 1–9.

[17] S. Haeri and L. Trajkovic, "Virtual network embedding via monte carlo tree search," *IEEE Trans. Cybern.*, vol. 48, no. 2, pp. 510–521, 2018.

[18] D. Pisinger and P. Toth, "Knapsack problems," in *Handbook of combinatorial optimization*. Springer, 1998, pp. 299–428.

[19] H. Yao, X. Chen, M. Li, P. Zhang, and L. Wang, "A novel reinforcement learning algorithm for virtual network embedding," *Neurocomputing*, vol. 284, pp. 1–9, 2018.

[20] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *17th European Conference on Machine Learning (ECML), Berlin, Germany*, Sep. 2006, pp. 282–293.

[21] S. James, G. D. Konidaris, and B. Rosman, "An analysis of monte carlo tree search," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, California, USA*, Feb. 2017, pp. 3576–3582.

[22] X. Ma, K. R. Driggs-Campbell, Z. Zhang, and M. J. Kochenderfer, "Monte carlo tree search for policy optimization," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI), Macao, China*, Aug. 2019, pp. 3116–3122.

[23] W. Wang, P. Hong, D. Lee, J. Pei, and L. Bo, "Virtual network forwarding graph embedding based on tabu search," in *9th International Conference on Wireless Communications and Signal Processing (WCSP), Nanjing, China*, Oct. 2017, pp. 1–6.