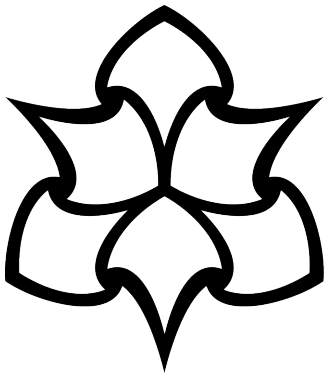

A Single-Player Monte Carlo Tree Search Method Combined with Node Importance for Virtual Network Embedding

Guangcong Zheng[1,*] · Cong Wang[1,*✉] · Weijie Shao[1] · Ying Yuan[1] · Zejie Tian[1] · Sancheng Peng[2] · Ali Kashif Bashir[3,4] · Shahid Mumtaz[5]

Abstract As a critical technology in network virtualization, virtual network embedding (VNE) focuses on how to allocate physical resources to virtual network requests efficiently. Because the VNE problem is NP-hard, most of the existing approaches are heuristic-based algorithms that tend to converge to a local optimal solution and have a low performance. In this paper, we propose a algorithm that combines the basic Monte Carlo tree search (MCTS) method with node importance to apply domain-specific knowledge. For a virtual network request, we first model the embedding process as a finite Markov decision process (MDP), where each virtual node is embedded in one state in the order of node importance that we design. The shortest-path algorithm is then applied to embed links in the terminal state and return the cost as a part of the reward. Due to the reward delay mechanism of the MDP, the result of link mapping can affect the action selected in the previous node-mapping stage, coordinating the two embedding stages. With node importance, domain-specific knowledge can be used in the Expansion and Simulation stages of MCTS to speed up the search and estimate the simulation value more accurately. The experimental results show that, compared with the existing classic algorithms, our proposed algorithm can improve the performance of VNE in terms of the average physical node utilization ratio, acceptance ratio, and long-term revenue-to-cost ratio.

Keywords Network virtualization · Virtual network embedding · Reinforcement learning · Markov decision process · Monte-Carlo tree search · Node ranking

1 Introduction

Over the past three decades, the Internet has achieved incredible success in supporting a variety of network technologies and distributed applications. However, due to its multi-provider nature, making changes to Internet architecture has become increasingly difficult. Hence, network virtualization has been proposed to allow multiple heterogeneous network architectures to coexist on a shared physical substrate network [1].

In network virtualization, the role of traditional Internet service providers (ISPs) is separated into two independent roles: that of infrastructure providers (InPs), who manage the physical infrastructure, and that of service providers (SPs), who provide virtual network (VN) services using resources from multiple InPs [2].

Efficient allocation and scheduling of physical resources among multiple VN requests (VNRs) is crucial to maximize the number of coexisting VNs, and to increase the utilization and revenue of InPs [3]. As shown in Fig. 1, the virtual network embedding (VNE) problem is to embed the nodes and links of VNs into the substrate network in a way that consumes the smallest amount of resources.

This work was supported by the National Natural Science Foundation of China under Grant Nos. 61702089, the basic scientific research operating fund of central universities under Grant No. N182304021, and the scientific research plan for institutions of higher learning of Hebei province under Grant No. ZD2019306.

✉Cong Wang
E-mail: congw@neuq.edu.cn

[*] These authors contributed equally to this work and should be considered co-first authors

[1] School of Computer and Communication Engineering, Northeastern University at Qinhuangdao, Qinhuangdao, Hebei 066004, China.

[2] Laboratory of Language Engineering and Computing, Guangdong University of Foreign Studies, Guangzhou, 510006, China.

[3] Department of Computing and Mathematics, Manchester Metropolitan University, UK.

[4] School of Electrical Engineering and Computer Science, National University of Science and Technology, Islamabad (NUST), Pakistan

[5] Instituto de Telecomunications (IT), Portugal.

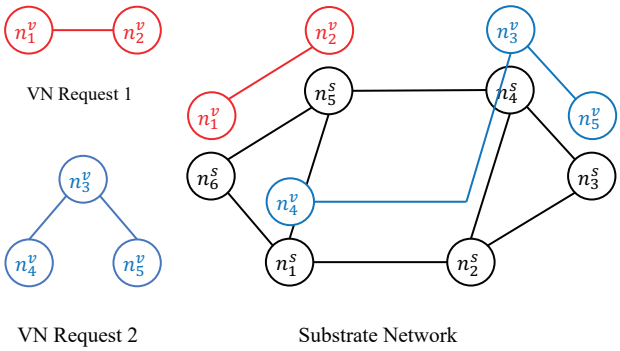

Fig. 1 Virtual network embedding

VNE can be performed using exact solutions such as integer linear programming (ILP) and mixed integer programming (MIP) [4,5]. However, solving a MIP is known to be an NP-hard problem [6]. Therefore, solution methods mainly focus on heuristic algorithms [7]. The process of VNE is usually divided into two stages: the node-mapping stage and the link-mapping stage. Most heuristic algorithms use node ranking-based approaches to solve the two stages separately [8–12]. Node importance metrics are usually used in the node-mapping stage to rank virtual and substrate nodes. Then, the Dijkstra or multi-commodity flow (MCF) algorithm is used in the link-mapping stage to embed links. However, such uncoordinated two-stage embedding methods do not take into account the relationship between the link-mapping stage and the node-mapping stage. Meanwhile, the performance of these algorithms relies heavily on the designed node importance metrics.

Reinforcement learning (RL) is an area of machine learning (ML) that enables self-improving agents to learn from interactions experienced in the environment to achieve a goal [13]. The RL problem is usually formulated as a Markov decision process, whose reward delay mechanism is suitable for the delay-sensitive problems [14] such as VNE. Since reinforcement learning is a type of unsupervised learning, we can set the profit of the VNE result as its reward without the need to manually mark large amounts of data. The environment of the VNE problem is fully observable, deterministic, and discrete, so it is easy for us to model the VNE environment. Instead of separately solving two embedding stages as most heuristic approaches do, we formulate a coordinated two-stage-based MDP to relate the two stages. As a methods of addressing MDPs with large state spaces, Monte Carlo tree search (MCTS) does not need an explicit evaluation function like other tree search methods, such as A* or IDA* [15]. It can use either light or heavy playouts. Instead of moving randomly in light playouts, domain-specific knowledge may be employed as heuristics in heavy playouts to influence the choice of moves [16].

At present, research using MCTS [17–19] to solve the VNE problem is still in the minority. We attempt to combine the basic MCTS with the domain-specific knowledge in the VNE problem, as we think that this useful knowledge will guide the search for an optimal direction. The node importance used in node ranking-based approaches can be an efficient way to incorporate the domain-specific knowledge of the VNE problem, and none of these approaches [17–19] have considered node importance. Therefore, in this paper, instead of testing to determine which node importance metric performs best, we focus on the problem of how to combine node importance with the basic MCTS. Two ways of utilizing node importance are proposed in our paper: determining the embedding order of virtual nodes before the MCTS and determining the probability of substrate nodes to be selected during the MCTS. For virtual nodes, a simple node importance metric is used to apply the knowledge of the local network resources. Specifically, the importance of a virtual node is defined as the sum of the node's CPU requests and its adjacent link bandwidth. This means that the agent will give priority to embedding the virtual nodes that are considered more valuable, e.g., the ones requesting more CPU and link bandwidth resources. This helps avoid additional time complexity due to the numerous embedding orders and can also speed up the search because the embedding process will fail earlier if there exists no feasible solution in current situation. We only calculate the virtual node importance for each virtual network once before starting the MCTS. For substrate nodes, a node importance metric called the minimum bandwidth consumption in theory is designed to apply the knowledge of the topology of the substrate network. If a selected substrate node leads to less bandwidth consumption, it is considered more valuable, and the agent should give priority to selecting it in the Expansion and Simulation stages of the MCTS.

The reward of the VNE problem is different from that of two-player go games such as chess. This is because that the VNE problem is not just about winning or losing. It is more appropriate to view it as a single-player game where a single VNE agent tries to maximize the revenue of embedding VNRs. The reward in the VNE problem is the revenue of a successful embedding, which may reach a large number instead of always lying in the range -1 or 1. Therefore, we propose applying the single-player MCTS instead of the basic MCTS. This is achieved by replacing the conventional UCT (Upper Confidence Bound 1 applied to trees) with the SP-UCT [20] (Single-player UCT) in Selection stage. The added variance of the simulation results in SP-UCT is expected to guide the agent to search for a better direction than the results of the UCT.

Then we will describe our approach to the VNE problem. We first transform the VNE problem into a model-based reinforcement learning (RL) problem by formulating the embedding process as a finite MDP, in which the agent operates in the modelled simulation environment to obtain rewards. In each state of the MDP, a substrate node is selected, and one virtual node is mapped onto it. In the terminal state, the shortest-path algorithm is applied to embed links, and it returns the link embedding cost as one part of the reward. Due to the reward delay mechanism of the MDP, the result of link mapping can affect the node mapping stage, coordinating the two embedding stages. This improves the poor performance caused by the separation of the two related embedding stages of the VNE problem in node ranking-based approaches. Before the MDP, we have to determine the embedding order of virtual nodes instead of using a random order. Otherwise, the time complexity will be too high. Therefore, node importance metrics are applied to rank virtual nodes in non-increasing order. This means that we will prioritize embedding virtual nodes that are considered more valuable. Then, a single-player MCTS method for the VNE problem (VNE-SPMCTS) is used to solve this coordinated two-stage-based MDP. Additionally, the node importance of substrate nodes is applied in our VNE-SPMCTS to expand the tree node according to the guidance of domain-specific knowledge of VNE and to estimate the values of the simulation more accurately. Our evaluation results are compared to those of one of the state-of-the-art VNE algorithms, named MaVEn-S [17], which uses a basic MCTS and the shortest-path link embedding method to solve the VNE problem.

The remainder of this paper is organized as follows: In section two, we review the related work on the VNE problem. A mathematical model for the VNE problem is established in section three, including the network model and problem formalization. In section four, a coordinated two-stage-based MDP for the VNE problem is proposed. In section 5, we propose a single-player Monte Carlo tree search (SP-MCTS) approach combined with the knowledge of node importance to solve the MDP of the VNE problem. In section 6, we verify and analyse the per-formance of our proposed algorithm. Finally, conclusions are drawn in section 7.

2 Related Work

The main purpose of VNE is to maximize resource utilization under the premise of limited physical network resources [3].

According to the mathematical models and methods used, the optimization of VNE can be classified into three categories: exact, heuristic, and meta-heuristic solutions [21].

An exact solution [6] is the optimal solution to the VNE problem. Concrete VNE exact solutions [4, 5] are usually formulated in terms of optimization theory, such as integer linear programming (ILP) and mixed integer programming (MIP). For example, Chowdhury *et al.* [4] formulated VNE as a MIP by introducing meta-nodes and relaxed the integer program to obtain an LP formulation that could be solved in polynomial time. Then, two new VN embedding algorithms called R-ViNE (randomized VN embedding) and D-ViNE (deterministic VNE embedding) were proposed. However, due to the high computational complexity, such an exact solution can not commonly be obtained in large-scale networks.

A heuristic solution method can find a good solution in a large-scale network, and it is a trade-off between efficiency and performance [7]. Most heuristic solutions [8–10] apply node importance metrics to rank nodes in the node-mapping stage and use the shortest-path or MCF algorithm in the link-mapping stage to embed links. Inspired by PageRank, used in the Google search engine, Xiang Cheng *et al.* [9] designed a topology-aware node-ranking algorithm named NodeRank, which is based on resource and topological attributes. Two VNE algorithms were proposed: RW-MaxMatch which maps virtual nodes to substrate nodes according to the ranking and uses the shortest-path or MCF algorithm to embed links, and RW-BFS which is a back-tracking VN embedding algorithm based on a breadth-first
search.

Meta-heuristic solutions such as particle swarm optimization [22], ant colony optimization [23], genetic algorithms [24], and simulated annealing algorithms [25] can find near-optimal solutions and improve performance by escaping from local optima. For example, the authors of [26] combined discrete particle swarm optimization (DPSO), simulated annealing algorithms, and taboo-search technology to solve the premature convergence problem. Then, they proposed a VNE algorithm based on hybrid particle swarm optimization.

In recent years, some works have begun to use reinforcement learning (RL) algorithms to solve the VNE problem [17–19, 27, 28]. In [27], a decentralized Q-learning based algorithm was proposed to approximate the state-action values, which are used to select actions to allocate substrate resources. At present, research using MCTS is still in the minority. Haeri *et al.* [17] used a basic MCTS to solve the MDP of the VNE problem. They proposed two algorithms called MaVEn-S and MaVEn-M [17], respectively. MaVEn-S employs the shortest-path algorithm in the link mapping stage while MaVEn-M uses the multi-commodity flow algorithm. Yao *et al.* [28] proposed an RL based dynamic attribute matrix representation (RDAM) algorithm for virtual network embedding, in which a substrate network is represented as a static matrix and can be dynamically updated.

Haipeng Yao *et al.* [19] combined MCTS with a policy network to make node mapping decisions. They trained a policy network with a historical embedding solution for VNRs, and then applied a policy gradient to achieve optimal embedding automatically. However, none of these researchers have combined MCTS with the knowledge of node importance, which is widely used in node ranking based approaches to rank nodes. A large amount of domain-specific knowledge can be applied to calculate the importance of virtual and substrate nodes. Therefore, an efficient way to apply domain-specific knowledge is to combine node importance with the basic MCTS. In this paper, we combine the basic MCTS with node importance metrics in both the Expansion and Simulation stages to improve the search efficiency.

3 Virtual Network Embedding Problem

In this section, a mathematical model is introduced for the VNE problem, including the basic network model, problem formalization, objective function, and performance metrics.

3.1 Network Model

3.1.1 Substrate Network

A substrate network (SN) is modelled as a weighted undirected graph $G^s=(N^s,E^s,A^s)$, where N^s is the set of substrate nodes, E^s is the set of substrate links, and A^s is the set of attributes of the substrate network.

In this paper, the dependencies such as memory of nodes, storage of nodes, and delay of links are ignored and only the CPU capacity and link bandwidth are considered. Each substrate node $n^s \in N^s$ has a CPU capacity $A^s_{cpu}(n^s)$ and each substrate link $e^s(n^s_i,n^s_j) \in E^s$ between the substrate node n^s_i and n^s_j has a bandwidth $A^s_{bw}(e^s)$. Let $P^s(n^s_i,n^s_j)$ denote the set of all the paths p^s between nodes n^s_i and n^s_j.

3.1.2 Virtual Network Request

Similar to a substrate network, a virtual network (VN) is modelled as $G^v=(N^v,E^v,A^v)$, where N^v is a set of virtual nodes, $A^v_{cpu}(n^v)$ denotes the CPU request of virtual node n^v, and $A^v_{bw}(e^v)$ denotes the bandwidth request of the virtual link e^v. In addition, a virtual network request (VNR) can be defined as $\phi(G^v,t_d) \in \Phi$, where t_d is the time duration that the VN requests to stay in the SN, which is called the lifetime.

3.2 VNE Problem Formalization

When a VNR arrives, the SN should decide whether to accept or reject it. If the VNR is accepted, the resources it requests should be allocated on the corresponding substrate nodes and paths that the VNR is mapped onto. If there are not sufficient physical resources available, the VNR should be rejected or postponed. If a VNR leaves the substrate network, the resources it occupied are released immediately.

The VNE problem can be defined as a mapping:

$$M : G^v(N^v,E^v,A^v) \rightarrow G^s(N^s,E^s,A^s), \tag{1}$$

which can be divided into the following two stages: the node-mapping stage M_N and the link-mapping stage M_E.

$$M = (M_N,M_E) \tag{2}$$

3.2.1 Node-mapping stage

The task of the node-mapping stage is to assign the nodes of the virtual network to the nodes of the substrate network. It is defined as a mapping:

$$\begin{aligned} &M_N : N^v \rightarrow N^s \\ &\forall n^v \in N^v, \exists n^s \in N^s, M_N(n^v) = n^s, \end{aligned} \tag{3}$$

and should meet the following constraint:

$$A^v_{cpu}(n^v) \leq R^s_{cpu}(n^s), \tag{4}$$

where $R^s_{cpu}(n^s)$ denotes the residual cpu resources of substrate node n^s.

For a VNR, different virtual nodes should be mapped onto different substrate nodes; this is defined as

$$M_N(n^v_i) \neq M_N(n^v_j), \forall n^v_i, n^v_j \in N^v, i \neq j. \tag{5}$$

3.2.2 Link-mapping stage

The task of the link-mapping stage is to map the links of a virtual network to the paths of the substrate network . It can be defined as a mapping:

$$\begin{aligned} &M_E : E^v \rightarrow P^s \\ &\forall e^v(n^v_i,n^v_j) \in E^v, \exists p^s \in P^s, M_E(e^v) = p^s \end{aligned} \tag{6}$$

and should meet the following constraint:

$$A^v_{bw}(e^v) \leq R^s_{bw}(e^s), \forall e^s \in p^s, \tag{7}$$

where p^s is the path between substrate node $M_N(n^v_i)$ and $M_N(n^v_j)$ and R_{bw} is the residual bandwidth of the substrate link e^s.

Since a virtual link can be embedded onto a substrate path p^s with several links, let $hops(p^s)$ denote the number of hops of the substrate path, which can be defined as

$$hops(M_E(e^v)) = hops(p^s) = |p^s|. \tag{8}$$

3.3 Objective Function

The revenue of a successfully mapped virtual network G^v is represented by the total resources it demands.

$$Revenue(G^v) = \sum_{n^v \in N^v} A^v_{cpu}(n^v) + \sum_{e^v \in E^v} A^v_{bw}(e^v) \tag{9}$$

The cost of the SN to sustain the virtual network G^v is defined as:

$$Cost(G^v) = \sum_{n^v \in N^v} A^v_{cpu}(n^v) + \sum_{e^v \in E^v} A^v_{bw}(e^v) \cdot hops(M_E(e^v)). \tag{10}$$

We apply the score function from MaVEn-S [17]. The score for a single VNR $\phi(G^v, t_d)$ is denoted by a function $Score()$.

$$Score(G^v) = \begin{cases} Revenue(G^v) - Cost(G^v) & \text{accepted} \\ -\infty & \text{rejected} \end{cases} \tag{11}$$

The goal of this paper is to maximize the above objective function in the long run.

3.4 Performance Metrics

In this subsection, some metrics are introduced, which are typically used to measure the performance of the VNE algorithms.

3.4.1 Average physical node utilization ratio (AUR)

For the entire SN, let the ratio between the total CPU resources used and the total CPU capacity of all physical nodes represent the efficiency of SN utilization.

$$AUR(G^s) = \frac{\sum_{n^s \in N^s} A^s_{cpu}(n^s) - R^s_{cpu}(n^s)}{\sum_{n^s \in N^s} A^s_{cpu}(n^s)} \times 100\% \tag{12}$$

3.4.2 Acceptance Ratio (AC)

Let the ratio between the number of accepted VNRs $|\Phi'|$ and the total number of arriving VNRs $|\Phi|$ be denoted as the acceptance ratio. It can also be seen as the probability of successfully embedding a VNR.

$$AC(\Phi) = \frac{|\Phi'|}{|\Phi|} \tag{13}$$

3.4.3 Long-term revenue to cost ratio (R/C)

The long-term R/C ratio [29] is usually used to quantify the performance of the algorithm, and it can be defined as the ratio between the long-term revenue and the long-term cost of the SN.

The long-term revenue is defined as

$$\lim_{T \to \infty} \sum_{t=0}^{T} Revenue(G^v). \tag{14}$$

The long-term cost is defined as

$$\lim_{T \to \infty} \sum_{t=0}^{T} Cost(G^v). \tag{15}$$

Thus the long-term R/C ratio is defined as

$$R/C = \lim_{T \to \infty} \frac{\sum_{t=0}^{T} Revenue(G^v)}{\sum_{t=0}^{T} Cost(G^v)}. \tag{16}$$

4 Formulate the VNE problem as a finite Markov decision process

In this section, we first introduce the finite Markov decision process. Then, we will formulate the VNE problem as a coordinated two-stage-based MDP.

4.1 Finite Markov decision process

The reinforcement learning problem is a framework for problems that learn from interactions to achieve goals. The decision-maker and learner together are called the agent. The things the agent interacts with, including everything outside it, are collectively called the environment.

A reinforcement learning task that satisfies the Markov property [13] (that the effects of an action taken in a state depend only on that state and not on the prior history) is called a Markov decision process. It is a discrete time stochastic control process. A finite MDP can be defined by a 5-tuple $(\mathcal{S}, \mathcal{A}, p(), \mathcal{R}, \gamma)$ where:

- $\mathcal{S}$ is a finite set of possible states, where $s_t \in \mathcal{S}$ is the state at time t
- $a_t \in \mathcal{A}(s_t)$, where $\mathcal{A}(s_t)$ is a finite set of available actions in state s_t
- $p(s'|s,a)$ is the state-transition probability to state s' from state s by taking action a
- $r_t \in \mathcal{R} \subset \mathbb{R}$, is a numerical reward
- γ is the discount factor for reward propagation and satisfies $0 \leq \gamma \leq 1$.

In a so-called episodic task, each *episode* ends in a terminal state. If the sequence of rewards received after time step t is denoted as r_{t+1}, r_{t+2}, r_{t+3}, and etc., the expected return G_t is defined by some specific functions of the reward sequence. In the simplest case, the return is the sum of all rewards accumulated so far and can be defined as

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k \cdot r_{t+k+1}, T \in \mathbb{N}, \tag{17}$$

where $T \in \mathbb{N}$ is the final time step. The discount factor γ is used so that the decision-maker tends to apply actions that can obtain rewards in the short-term future rather than the long-term future.

Given any state s and action a, the probability of each possible pair of a next state s' and reward r is denoted by

$$p(s',r|s,a)=Pr\{s_{t+1}=s',r_{t+1}=r|s_t=s,a_t=a\}. \tag{18}$$

Given the dynamics as specified by (18), the rest of the information about the *environment* can be calculated, such as the expected rewards for state-action pairs:

$$\begin{aligned} r(s,a) &= \mathbb{E}[r_{t+1}|s_t=s,a_t=a] \\ &= \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p(s',r|s,a), \end{aligned} \tag{19}$$

the state-transition probabilities:

$$\begin{aligned} p(s'|s,a) &= Pr\{s_{t+1}=s'|s_t=s,a_t=a\} \\ &= \sum_{r\in\mathcal{R}} p(s',r|s,a), \end{aligned} \tag{20}$$

and the expected rewards for state-action-next-state triples:

$$\begin{aligned} r(s,a,s') &= \mathbb{E}[r_{t+1}|s_t=s,a_t=a,s_{t+1}=s'] \\ &= \frac{\sum_{r\in\mathbb{R}} rp(s',r|s,a)}{p(s'|s,a)}. \end{aligned} \tag{21}$$

4.2 MDP of coordinated two-stage VNE

In this subsection, we will formulate a coordinated two-stage VNE process as a finite MDP. The so-called coordinated two-stage embedding is mainly achieved by the reward delay mechanism of the MDP. The reward in the node-mapping stage will not be returned until the completion of the link-mapping stage, that is, until the entire mapping is completed. In this way, the mapping result of the link-mapping stage can affect the action selection process in the previous node mapping stage.

Take the decision-making of VNE as the *agent* and the substrate network, its resources, and its VNRs as the *environment*. The VNE agent aims to satisfy a VNR by embedding its virtual network into the substrate network. In the MDP of the VNE problem, the state s at time step t can be defined as

$$s_t = (M_N^t, N_t^v, N_t^s), \tag{22}$$

where M_N^t is the node mapping function at time step t, N_t^v is the ordered set of residual virtual nodes at time t that are requesting to be embedded, and N_t^s is the set of residual substrate nodes at time t that are available to be embedded into. At the first time step t_0, the initial mapping function M_N^0 is arbitrary.

Let T denote the final time step, where the value of T is equal to the number of virtual nodes. At each time step $t \in [0, T-1]$, the VNE agent selects a substrate node n_i^s that satisfies the node mapping constraint in (5) and then takes the action $a(n_t^v, n_i^s)$, where $a(n_t^v, n_i^s)$ means that the $t+1$-th virtual node n_t^v in the ordered set N_t^v is mapped onto the substrate node n_i^s. After the virtual node n_t^v is successfully embedded, the game moves from state s_t to a new state s_{t+1}, which is defined as

$$\begin{aligned} N_{t+1}^v &= N_t^v \setminus \{n_t^v\} \\ N_{t+1}^s &= N_t^s \setminus \{n_i^s\} \\ M_N^{t+1} &= M_N^t, M_N^{t+1}(n_t^v) = n_i^s \\ (M_N^t, N_t^v, N_t^s) &\Rightarrow (M_N^{t+1}, N_{t+1}^v, N_{t+1}^s). \end{aligned} \tag{23}$$

However, the successful mapping of a single virtual node cannot generate a reward, which means

$$r(s_t, a, s_{t+1}) = 0, \forall t \in [0, T-2]. \tag{24}$$

If state s_0 is not able to arrive at the terminal state s_T, i.e., some virtual nodes cannot be embedded, the node-mapping stage fails and leads to the rejection of the VNR.

If state s_0 leads to a terminal state s_T, the node-mapping stage succeeds. Then, the VNE process enters the link-mapping stage and uses the shortest-path algorithm to embed links. If the link-mapping stage also succeeds, the VNR is accepted. Otherwise, it is rejected or postponed. Ultimately, the VNE agent returns the reward r calculated by (11) according to the current embedding result and updates the expected reward of previous states.

$$r(s_t, a, s_{t+1}) = Score(G^v), t = T-1 \tag{25}$$

As shown in Fig. 2, there is a finite and coordinated two-stage based MDP that represents the process of embedding VNR 1 into the substrate network in Fig. 1. Assume that the order of virtual nodes is fixed as $\{n_1^v, n_2^v\}$. In the initial state s_0, the agent randomly chooses an action a from the actions set $\mathcal{A}(s)$, such as $a_0(n_1^v, n_6^s)$. Then, state s_0 goes to the next

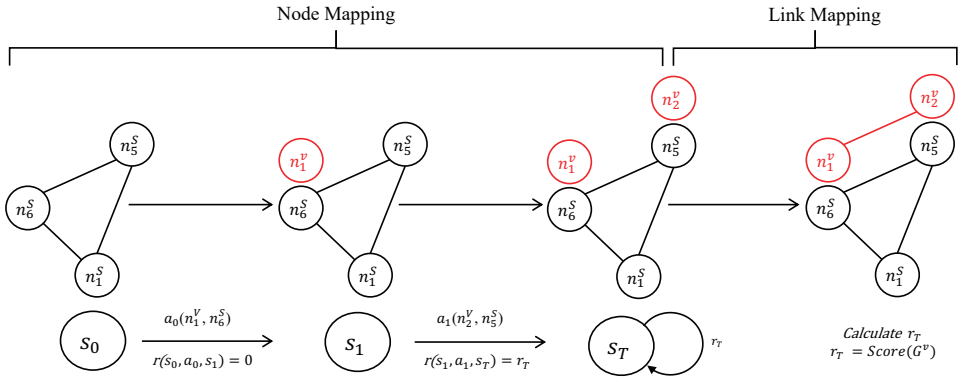

Fig. 2 A finite and coordinated two-stage based Markov decision process of VNE

state s_1, where the virtual node n_1^v has already been embedded into the substrate node n_6^s. The transition from state s_0 to state s_1 represents the successful embedding of virtual node n_1^v and obtains a reward $r(s_0, a_0, s_1)$=0. Then, in state s_1, the agent applies action $a_1(n_2^v, n_5^s)$ to embed the virtual node n_2^v into the substrate node n_5^s, and state s_1 goes to the terminal state s_T. The arrival at s_T represents the success of the node-mapping stage, and then the embedding process moves to the link mapping stage. The cost of this solution is calculated by a shortest-path algorithm and is returned as a part of the reward $r(s_1, a_1, s_T)$=$Score(G^V)$. The simulation value is then used to update the evaluation value of previous states.

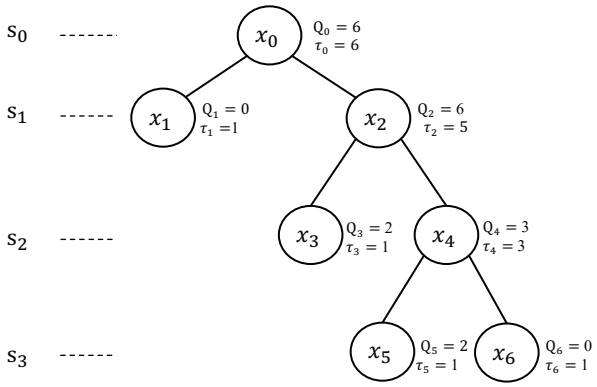

Fig. 3 Monte Carlo tree of the VNE problem

5 Single-Player Monte Carlo Tree Search Combined with Node Importance

At present, research using MCTS to solve the VNE problem is still in the minority. Among these early studies [17–19], none of them have combined MCTS with the knowledge of node importance, which is widely used in node ranking-based approaches to rank nodes. A large amount of domain-specific knowledge can be applied to calculate the importance of virtual and substrate nodes. Therefore, an efficient way to apply domain specific-knowledge is to combine node importance with the basic MCTS. In this section, a basic Monte Carlo tree search method is constructed to demonstrate the general process of how MCTS solve the MDP of VNE. Then, a refined Single-Player Monte Carlo tree search combined with the node importance named VNE-SPMCTS is proposed to achieve better performance. At the end of this section, we present the pseudocodes of our proposed VNE-SPMCTS.

5.1 Basic MCTS to address the MDP

MCTS is a best-first search algorithm which iteratively employs stochastic simulations to achieve efficient searching results in a tree. As shown in Fig. 3, the depth of a Monte Carlo tree (MCT) is equivalent to the value of the final time step T in an MDP.

Let x_t denote the tree node at the t-th level of an MCT. It can be defined as a 3-tuple $x_t = (s_t, \tau, Q)$, where s_t denotes the state at time t in the MDP, τ denotes the number of times that the node itself and its children have been visited (i.e. number of iterations on this subtree), and Q is the sum of the simulation values accumulated so far. In each search, the best child node of x_0 will be selected after I iterations. In each iteration of a search, there are four stages: **Selection**, **Expansion**, **Simulation**, and **Backpropagation**.

5.1.1 Selection

Generally, in an MCT, a tree node is expandable if it represents a nonterminal state and has unvisited children (i.e., it is unexpanded). Starting at the root node x_0, a child selection

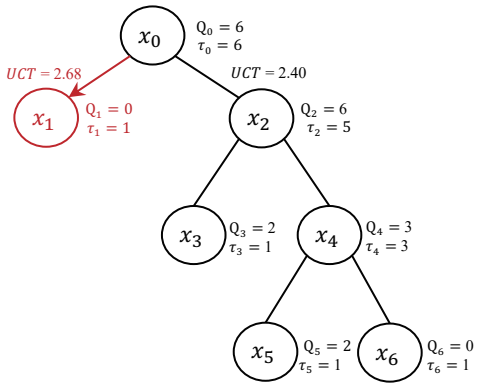

Fig. 4 Selection stage

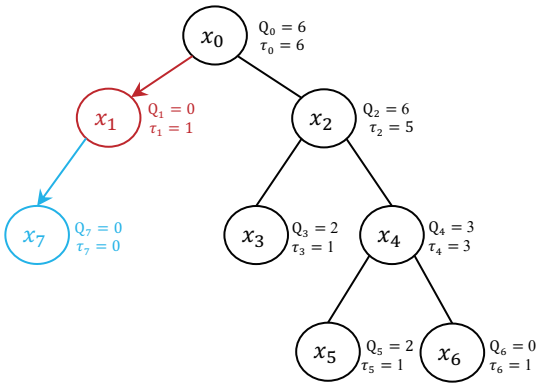

Fig. 5 Expansion stage

strategy is recursively applied to descend through the tree until the most urgent expandable node is reached, in a way that balances between exploitation and exploration.

- **Exploitation** aims to select the node that leads to the best results obtained so far.
- **Exploration** aims to explore relatively less promising actions due to the uncertainty of the evaluation, since these actions may later turn out to be better.

The first formula for balancing exploitation and exploration in a game, propounded by Kocsis *et al.* [30], was called Upper Confidence Bound 1 applied to trees (UCT).

$$\frac{Q_i}{\tau_i} + C\sqrt{\frac{\ln \tau'}{\tau_i}} \tag{26}$$

- τ_i denotes the number of simulations for x_i.
- $\frac{Q_i}{\tau_i}$ denotes the average value of the simulations.
- τ' denotes the number of simulations for the father node of x_i.
- C is the exploration parameter which theoretically equals to $\sqrt{2}$ and in practice is usually assigned empirically.

A motivating example is shown in Fig. 4 to clarify the mechanism of the Selection stage. The UCT values of node x_1 and x_2 are 2.68 and 2.40 respectively according to (26). Thus, in the Selection stage of this iteration, the algorithm selects x_1 as the most urgent expandable node in state s_1.

5.1.2 Expansion

After the most urgent expandable node is determined in Selection stage, the Expansion stage starts. The algorithm randomly creates one or more child nodes and adds them to the MCT.

As shown in Fig. 5, in Expansion stage, the algorithm randomly chooses an action a from the untried actions set

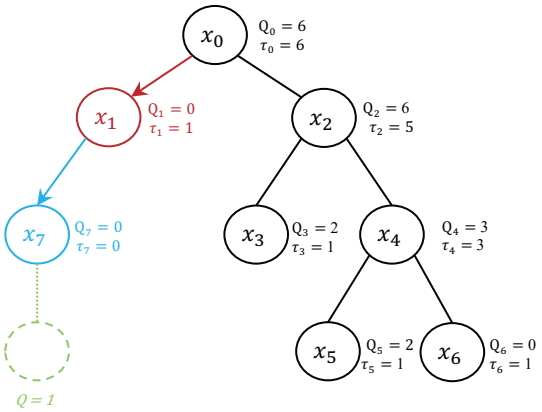

Fig. 6 Simulation stagee

$\mathcal{A}(s_1)$, where s_1 is the state of tree node x_1. Then, s_1 changes to s_2 after applying action a. As a result, a new node x_7 with state s_2 is generated and added to the MCT as the child of x_1.

5.1.3 Simulation

In the Simulation stage, a simulation is carried out from the new node expanded in the Expansion stage until the end of game according to the simulation policy (often using a random rollout) to produce a simulation value Q. With a large number of simulations, MCTS is able to calculate the expected estimate of each state.

As in the example shown in Fig. 6, a simulation value $Q = 1$ is generated. This simulation value corresponds to the reward function in the VNE problem.

5.1.4 Backpropagation

In Backpropagation stage, the simulation result Q is propagated backward to update the states of the nodes that are in the path from the newly expanded node to the root node. At

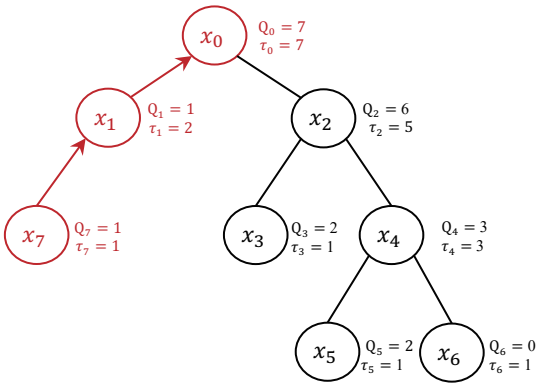

Fig. 7 Backpropagation stagee

the same time, the visit times τ of nodes in this path are also incremented. The updated values will be used in the next iteration to achieve more accurate estimation so as to gain a better embedding result.

As shown in Fig. 7, the new information including the simulation result $Q = 1$ and the increment of the visit times, is added to all nodes on the path from x_7 to x_0. Thus, the UCT values of these nodes are updated for the next iteration.

5.2 SP-UCT in the Selection stage

Notwithstanding the abovementioned basic MCTS, the VNE problem is different from the normal two-player go-game [31], in which the basic MCTS performs well. The VNE problem is a single-player game, in which the VNE agent attempts to maximize the Score() function mentioned in (11). The main distinction between these types of games lies in the range of values. In two-player games, the result of a game is denoted by {loss, draw, win} and has the value {-1,0,1}. The average score of a node always falls in [-1,1]. However, in single-player games, an arbitrary score can be achieved that is not in a pre-set interval by definition. For example, the score function of VNE mentioned in (11) falls in $(-\infty, 0]$. Therefore, specifically so that MCTS can address the VNE problem, we employ a variant UCT called SP-UCT [20].

$$\overline{q} + C\sqrt{\frac{2\ln \tau'}{\tau_i}} + \sqrt{\frac{\sum (q-\overline{q})^2 + D}{\tau_i}} \tag{27}$$

The first two terms come from the original UCT formula mentioned in (26) to give an upper confidence bound for the average game value $\overline{q}$, where τ_i denotes the number of times that the x_i is visited and τ' denotes the number of times that the father of x_i is visited. For the VNE problem, the SP-UCT adds a third term to represent a possible deviation of the child nodes. The third term can be represented as $\sqrt{\sigma^2 + \frac{D}{\tau_i}}$, where $\sigma^2 = \frac{\sum (q-\overline{q})^2}{\tau_i}$ represents the variance of the simulation results achieved so far in x_i. The constant D is added to ensure that the nodes that have rarely been explored are considered to be uncertain.

5.3 Combining MCTS with node importance

By applying the domain-specific knowledge of VNE, we expect an improvement on both accuracy and search efficiency. As in previous studies [8–12], applying node importance metrics is an effective way to combine models with domain-specific knowledge of the VNE problem. Therefore, in this section, we aim to combine the basic MCTS with node importance. Two ways of utilizing node importance have been proposed: determining the embedding order of virtual nodes before the MCTS and determining the probability of substrate nodes to be selected during the MCTS.

5.3.1 Node importance metric for ranking virtual nodes before the MCTS

Before using MCTS to solve the MDP, we have to fix the order of the virtual nodes to be embedded. Otherwise, the time complexity will be undesirably high. Instead of randomly determining the order, we use the node importance to rank the virtual nodes in a non-increasing order. This means that the agent will prioritize embedding virtual nodes that are considered more valuable. Inspired by the minimum remaining values (MRV) heuristic in CSPs(constraint satisfaction problems), the virtual node that should be embedded first is the one with the fewest legal values, i.e., the node that requires a large number of resources. Since a node that requires more resources will have fewer available substrate nodes for embedding, the search will end sooner if it is not determined to find a feasible solution. This helps to speed up the search. In this paper, the importance of a virtual node is related to the amount of resources it demands and is defined as the sum of its CPU demand and its neighboring link resources. For example, the node importance of virtual node n_i^v is defined as

$$I(n_i^v) = A_{cpu}^v(n_i^v) + \sum A_{bw}^v(e^v(n_i^v, n_j^v)), \tag{28}$$

where $e^v(n_i^v, n_j^v)$ is the edge between the virtual node n_i^v and its neighbouring node n_j^v.

As our work mainly focuses on how to combine node importance with MCTS, a simple node importance metric for virtual nodes is designed, which considers the knowledge of local network resources. The node importance $I(N^v)$

is calculated for ranking virtual nodes. In contrast to methods that rank substrate nodes before the search, we do not apply "large-to-large, small-to-small" embedding like the node ranking method [9] does. In our method, the embedding step is formulated as a finite MDP and is performed using MCTS. The ranking of virtual nodes in a search method can be likened to the ordering of variables in CSPs, which is used to speed up the search process.

5.3.2 Node importance metric for selecting substrate nodes during the MCTS

In practice, it is profitable to reduce the overall resource consumption by assigning devices under the same router to the virtual network. Based on this prior experience, a new node importance metric is designed.

When deciding which substrate node n_i^s to use for embedding the first virtual node n_t^v $(t=0)$ of a virtual network, we use the node importance metric $I(n^s)$, which is the sum of the node's remaining CPU capacity and its neighboring link resources. Specifically, the node importance of substrate node n_i^s is defined as

$$I(n_i^s) = A_{cpu}^s(n_i^s) + \sum A_{bw}^s(e^s(n_i^s, n_j^s)), \tag{29}$$

where $e^s(n_i^s, n_j^s)$ is the edge between substrate node n_i^s and its neighbouring node n_j^s. This policy will help the agent avoid wasting computation on trying those substrate nodes that may not provide enough resources for the subsequent link-embedding stage.

When deciding which substrate node n_i^s to use for embedding the virtual node n_t^v $(1 \le t < |N^v|)$, the node importance metric $I(n^s)$ is defined as

$$\begin{aligned} I(n_i^s) &= \frac{1}{H(n_i^s)} \\ H(n_i^s) &= \sum_{j=0}^{t-1} hops(p(n_i^s, M_N(n_j^v))) \times A_{bw}^v(e^v(n_t^v, n_j^v)), \end{aligned} \tag{30}$$

where $A_{bw}^v(e^v(n_t^v, n_j^v))$ denotes the bandwidth of virtual link between the virtual nodes n_t^v and n_j^v, $p(n_i^s, M_N(n_j^v))$ denotes the shortest path between the substrate nodes n_i^s and $M_N(n_j^v)$, $hops(p(n_i^s, M_N(n_j^v)))$ denotes the number of hops of this shortest path, and $H(n_i^s)$ denotes the minimum bandwidth consumption if the substrate node n_i^s is selected for embedding the virtual node n_t^v.

The hops of the shortest path between each pair of substrate nodes are relative to the topology of the substrate network only and can be calculated by the Floyd [32] algorithm for one time after reading the initial substrate network. Thus, the calculation of the substrate node importance will not lead to a large increase in time complexity.

After the computation of the node importance for each substrate node n_i^s is finished, the VNE agent uses it to decide on the best action $a(n_t^v, n_i^s)$. In each state s at time t, the available action $a(n_t^v, n_i^s)$ is the action that maps the virtual node n_t^v onto substrate node n_i^s under the premise that the action satisfies the embedding constraints mentioned in (5). Let $\sigma(a)$ denote the priority of a, which corresponds to our designed node importance metrics I for substrate nodes. It can be defined as

$$\sigma(a(n_t^v, n_i^s)) = \frac{I(n_i^s)}{I(n_{min}^s)} \ge 1, \tag{31}$$

where $n_{min}^s \in N^s$ denotes the available substrate node that has the smallest node importance; this node satisfies

$$I(n_{min}^s) = \min(I(N^s)). \tag{32}$$

The purpose of VNE is to reduce the consumption of the total bandwidth cost. The node importance for substrate nodes can be applied in the Expansion or Simulation stage of MCTS. In the Expansion stage, the node importance guides the agent to apply the action (expand the tree node) that may lead to less bandwidth consumption. In the Simulation stage, the agent tends to estimate the expected value positively. This can speed up the tree search and accelerate the process of finding a near-optimal solution.

However, unlike most of the existing node ranking-based approaches [8–10] that use deterministic "large-to-large and small-to-small" mapping, the node importance metrics of substrate nodes in this paper are used as probabilities in MCTS. Let $P(a)$ denote the probability of selecting the available action $a(n_t^v, n_i^s)$ at time t. It can be defined as

$$P(a_i) = \frac{\sigma(a_i)^\alpha}{\sum_{a \in \mathcal{A}(s)} \sigma(a)^\alpha}, \forall a_i \in \mathcal{A}(s), \tag{33}$$

where α is a scale factor designed to adjust the gap between different node importance values. If $\alpha > 1$, it enlarges the gap in the priority of action between different node importance values. If $\alpha = 1$, it does not adjust the gap. If $0 < \alpha < 1$, it reduces the gap. Clearly, $P(a)$ is a function calculated from $\sigma(a)$. It represents the probability of selecting action a according to its priority.

In this way, our algorithm can not only use the domain-specific knowledge of the node importance to find a near-optimal solution but can also use the characteristics of MCTS to avoid local optima.

5.4 Algorithm

Since the environment of VNE is fully observable, deterministic, and discrete, the VNE environment is easy to model. Thus, our proposed algorithm is a model-based reinforcement learning approach, in which we formulate the VNE

problem to create a simulation environment so that we can run our embedding in this virtual environment to obtain rewards. With the modelled environment, the agent can imagine the future through numerous simulations instead of running in the real world. This allows our agent to learn much more quickly and achieve better results. Our proposed VNE-SPMCTS is also an on-policy RL approach, in which the agent learns while acting. Given a substrate network and a virtual network request, the agent tries to embed the virtual network into the substrate network in an optimal way. First, the agent builds a Monte Carlo tree by creating the root node using the current condition of the substrate network. For each substrate network whose condition is different from that of the previous substrate network, we create a new Monte Carlo Tree. In other words, an MCT is only designed and trained to solve the problem of embedding a specific virtual network into the substrate network in a specific condition. However, even in this case, our algorithm still has better performance than traditional node ranking-based methods. Instead of finding one self-considered optimal solution as traditional node ranking-based algorithms do, MCTS-based methods search among several feasible solutions to find a near-optimal solution.

Algorithm 1 VNE-SPMCTS

Require: a substrate network and a virtual network request
Ensure: a near-optimal solution of VNE
1: sort the virtual nodes N^v according to I(N^v)
2: create the initial state s
3: set the iterations number I for each tree search
4: **while** s is not the terminal state **do**
5: $s' \leftarrow$ the best next state calculated by SP-MCTS(s, I)
6: $s \leftarrow s'$
7: **end while**
8: $M_N \leftarrow$ the node mapping result in the terminal state
9: $M_E \leftarrow$ the link mapping result obtained by Dijkstra
10: **return** (M_N, M_E)

The pesudocode in Algorithm 1 outlines the details of our proposed VNE-SPMCTS. Given a substrate network $G^s=(N^s, E^s, A^s)$ and a virtual network request $\phi(G^v, t_d)$, the VNE-SPMCTS is able to allocate the resources of the substrate network G^s to the requested virtual network $G^v=(N^v, E^v, A^v)$ for a time duration of t_d. The output of VNE-SPMCTS is a near-optimal embedding solution represented as a mapping function $M=(M_N, M_E)$, where M_N denotes the node mapping solution and M_E denotes the link mapping solution.

First, the virtual nodes n^v are sorted according to the node importance $I(n^v)$ in (28). The node importance will decide which is the best virtual node to embed first and will guide the tree to grow more towards proper branches with solutions. The task of the Monte Carlo tree search method is to select the best action a under the current state s. Therefore, in each state s_t of the MDP formulated in section 4.2, SP-MCTS shown in Algorithm 2 is used to build a Monte Carlo tree, and the agent performs I searches in this tree to decide the best action $a(n_t^v, n_i^s)$, where $a(n_t^v, n_i^s)$ means embedding the virtual node n_t^v into substrate node n_i^s. The performance of our algorithm can be improved by increasing the number of searches I as well as the time consumed. After applying the best action, the game moves from state s_t to s_{t+1}. The algorithm repeats the above operations until the state s_t reaches the terminal state s_T, which represents the success of the node mapping stage. Then, the shortest-path algorithm Dijkstra is applied to finish the link mapping, and to store the link mapping solution in M_E. Finally, the near-optimal embedding solution (M_N, M_E) is returned as the output of our proposed VNE-SPMCTS algorithm.

The pseudocode in Algorithm 2 outlines a single-player Monte Carlo tree search method, whose task is to decide the best action a under the given state s_t. First, the root node of a Monte Carlo tree is created with the given state s_t and I searches will be performed in this tree.

Algorithm 2 SP-MCTS(s, I)

Require: s is the current state in the MDP of VNE
Ensure: the best next state s'
1: create a tree node *root* with state s
2: **while** $I > 0$ **do**
3: *node* $\leftarrow$ Selection(*root*)
4: **if** *node* is expandable **then**
5: *node* = Expansion(*node*)
6: **end if**
7: *reward* $\leftarrow$ Simulation(*node*)
8: BackPropagation(*node*, *reward*)
9: $I \leftarrow I - 1$
10: **end while**
11: Among all the states of *root*'s child nodes, choose the state with the highest exploitation value as the best next state s'
12: **return** s'

Next, we step into the details of one search. Each search in an MCT consists of four parts: Selection with Algorithm 3, Expansion with Algorithm 4, Simulation with Algorithm 5, and Back-propagation with Algorithm 6.

Algorithm 3 Selection(x)

Require: the root node x
Ensure: the most expandable node
1: **while** tree node x is fully expanded **do**
2: x' select the best child node according to UCT(x) or SP-UCT(x)
3: $x \leftarrow x'$
4: **end while**
5: **return** x

As shown in Algorithm 3, the agent traverses the tree to find the best child node, where 'best' is defined by the

Single-Player Upper Confidence Bound applied to trees (SP-UCT) in (27).

Algorithm 4 Expansion(x)

Require: the most expandable node x
Ensure: x or a leaf node x' expanded from x
1: $s \leftarrow$ the state in node x
2: $\mathcal{A}(s) \leftarrow$ the untried action set in state x
3: $a \leftarrow$ an action from $\mathcal{A}(s)$ according to $I(a)$
4: $x' \leftarrow$ Move(x, a)
5: add x' into the set of x's child nodes
6: remove action a from $\mathcal{A}(s)$
7: **return** x'

As shown in Algorithm 4, the leaf node x selected in the Selection stage is not immediately expanded but waits for the number of visit $x.\tau$ to reach a certain number (i.e., 5). This avoids generating too many branches and decreasing the search attention. At the same time, the evaluation of a leaf node is more accurate when it is expanded. This seems to be an implementation trick, but it is of great importance for reaching a good solution. When the number of visits to the selected node x is large enough, the agent calculates the set of available actions $A(s)$ under the state of node x. Then, an action $a(n^v, n^s)$ from the actions set $A(s)$ is chosen according to the node importance $I(n^s)$ we designed in (29). After that, the agent uses $Move(x,a)$ to apply action $a(n^v, n^s)$ to the state s of node x and obtain a expanded node x'. The expanded node x' is then returned as the output of the Expansion stage.

Algorithm 5 Simulation(x)

Require: a leaf node x that is generated in Expansion stage
Ensure: the expected evaluation of the state in node x
1: **while** $x.state$ is not the terminal state **do**
2: **if** there is no available action in $x.state$ **then**
3: **return** -∞
4: **else**
5: $s \leftarrow$ the state in node x
6: $\mathcal{A}(s) \leftarrow$ the untried action set in state x
7: $a \leftarrow$ an action from $\mathcal{A}(s)$ according to $I(a)$
8: $x \leftarrow$ Move(x, a)
9: **end if**
10: **end while**
11: $s \leftarrow$ the state of node x
12: $r \leftarrow$ evaluateReward(s)
13: **return** r

As shown in Algorithm 5, numerous simulations from the current state s of node x to the terminal state are run in the modelled environment. Then, the agent obtains the reward r as the output of the Simulation stage, which is related to the embedding cost in the link mapping stage.

Back-propagation, shown in Algorithm 6, uses the simulation reward r to update the estimation values of the states

Algorithm 6 Back-propagation

Require: the leaf node x and its reward r
Ensure: None
1: **while** x is valid **do**
2: $x.\tau$ += 1
3: add r into $x.Q$
4: $x \leftarrow$ the parent node of x
5: **end while**

that are on the path from the leaf node to the root node. Thus, the embedding cost in the link mapping stage can affect the action selection in the previous node-mapping stage. This is the meaning of a coordinated two-stage-based algorithm means.

Assume that the substrate network has n nodes, and the current VNR needs m virtual links. In the node mapping stage, most node ranking algorithms first compute the importance of each substrate node and virtual node and then rank the nodes separately. The time complexity of ranking process is $O(n \lg n)$. According to the ranking result, these algorithms embed nodes sequentially. In the link mapping stage, the algorithms employ $Dijsktra$ algorithm m times, reaching a time complexity of $O(nm \lg n)$. Thus, a feasible solution is obtained. The overall time complexity of these node ranking algorithms is polynomial.

Compared to node-ranking algorithms that obtain only one relatively good solution for a VNR, our VNE-SPMCTS can obtain I feasible solutions for one node through I iterations (I is a hyper-parameter that is pre-set manually). The best one out of the I solutions is adopted. For a VN with n nodes, the time complexity is $O(Inm \lg n)$.

Clearly, these node-ranking algorithms are approximately I times faster than our algorithms, but our algorithm can obtain much better solutions, which is worthwhile in practice.

6 Performance Evaluation

To verify the performance of our algorithm VNE-SPMCTS proposed in this paper, our algorithm is compared with two other VNE algorithms: VNE-UEPSO [33] and MaVEn-S [17].

VNE-UEPSO is a typical meta-heuristic algorithm that establishes an ILP model for the VN embedding problem in which path splitting is unsupported, and it uses a unified enhanced particle swarm optimization (UEPSO) algorithm to solve the model. The UEPSO is a variant of the discrete particle swarm optimization with a biased local selection strategy for position initialization and update.

MaVEn-S is an MCTS based algorithm that formulates the VNE process as a finite MDP. However, in MaVEn-S, virtual nodes are in a random order and a basic MCTS without any domain-specific knowledge is introduced to solve the MDP.

6.1 Simulation Environment

The simulations are performed on two Intel Xeon E5-2640V4 CPUs with 64 GB RAM. All algorithms are implemented in CloudSim 3.0.3 [34], which is a famous simulation platform for cloud computing. A topology generator based on Java is pre-programmed to randomly generate the topology of the network. Most of the parameters are shown in Table. 1.

6.1.1 Substrate Network

The substrate topology has 64 nodes, and the CPU capacity of each node is 10000 units. The bandwidth resource of each substrate link is fixed as 10000 units. The link connectivity of the substrate topology is set to 0.1.

6.1.2 Virtual Network

For each virtual topology, the connectivity probability is set to 0.5, the number of virtual nodes is uniformly distributed between 4 and 10, the bandwidth of each virtual link is uniformly distributed between 500 and 5000 units, and the CPU request of each virtual node is uniformly distributed between 200 and 2000 units. For each VNR, its lifetime is uniformly distributed between 200 and 1000 time units in CloudSim.

6.1.3 Simulation Procedure

The experiment is simulated in a total of 20000 time units, where a time unit is determined by the simulated clock in CloudSim 3.0. The number of VNRs that arrive per 50 time unit is randomly uniform between 3 and 5. Then the agent applies the algorithm to embed or reject each VNR. The lifetime of a virtual network is randomly uniform between 200 and 1000. To guarantee fairness, for all algorithms, we use the same sequence of VNRs. The acceptance ratio (13), revenue-to-cost ratio (16), and average node utilization ratio (12) are stored per 1000 time units to evaluate the performance of the three algorithms.

In VNE-UEPSO [33], the size of the particle population is set to 30. The stopping criterion is that either the number of iterative rounds exceeds 30 iterations or the global optimal position has not changed during the previous eight rounds. The setting of the other of parameters in our evaluation are shown in Table. 1.

Table 1 Parameters for simulation

Parameter Item	Value
The total number of time units for simulation	20000
The number of VNRs that arrive per 50 time units	3-5
The number of nodes in the substrate network	64
The number of nodes in each virtual network	4-10
The connectivity probability of the substrate network	0.1
The connectivity probability of the virtual network	0.5
The cpu capacity of substrate nodes	10000
The cpu capacity of virtual nodes	200-2000
The bandwidth capacity of substrate links	10000
The bandwidth capacity of virtual links	500-5000
The lifetime of a virtual network request	200-1000
C in SP-UCT	10000
D in SP-UCT	10000
The number of iterations per MCTS	15

6.2 Experiments during time periods

In the simulation, we verify the performance of the three algorithms with the typical VNE performance metrics discussed in Section III. The number of iterations for the MCTS-based method is fixed at 15. Fig. 8 shows the average utilization ratio of all substrate nodes calculated by (12). Fig. 9 shows the acceptance ratio of all substrate nodes according to (13). Fig. 10 shows the long-term revenue to cost ratio according to (16). Our results show that the MCTS based

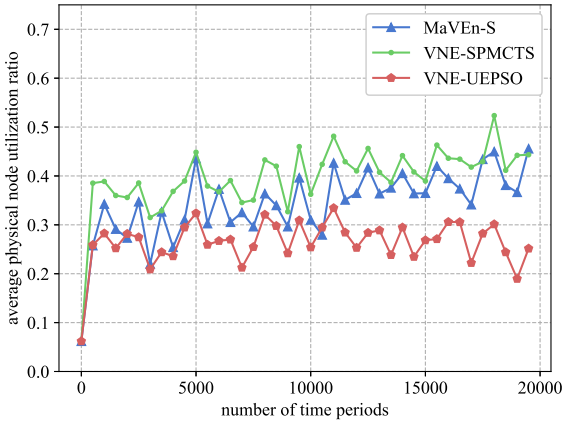

Fig. 8 Average physical node utilization ratio (AUR) over the time periods

algorithms MaVEn-S [17] and our VNE-SPMCTS perform much better than the DPSO based VNE-UEPSO in terms of the average physical node utilization ratio, acceptance ratio, and long-term revenue to cost ratio. The main reason is that the VNE process of MCTS-based algorithms is formulated as a coordinated two-stage-based MDP, which considers the connection between the separate node-mapping stage and link-mapping stage. Due to the introduction of the SP-UCT mechanism in the Selection stage and the node importance metrics used in the Expansion and Simulation stages of MCTS, our proposed VNE-SPMCTS algorithm can utilize the domain-specific knowledge and thus can further improve the embedding performance.

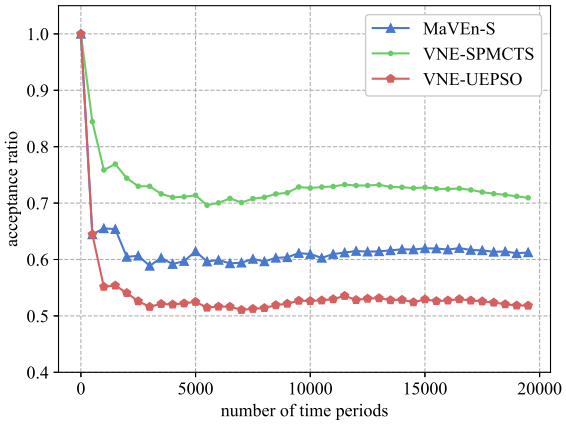

Fig. 9 Acceptance ratio over the time periods

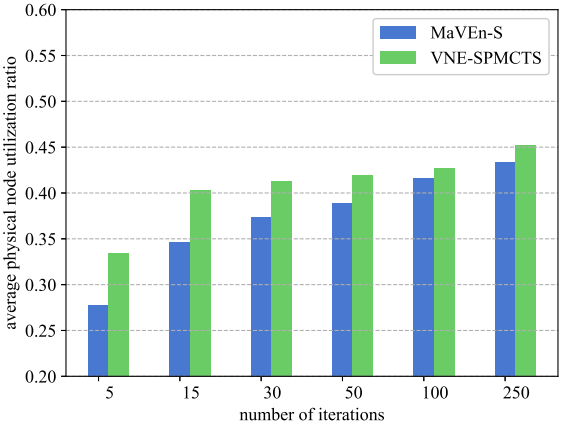

Fig. 11 Average physical node utilization ratio (AUR) under different numbers of iterations

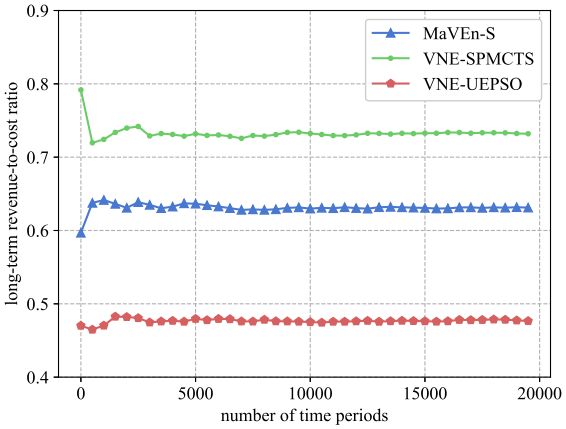

Fig. 10 Long-term revenue-to-cost ratio (R/C) over the time periods

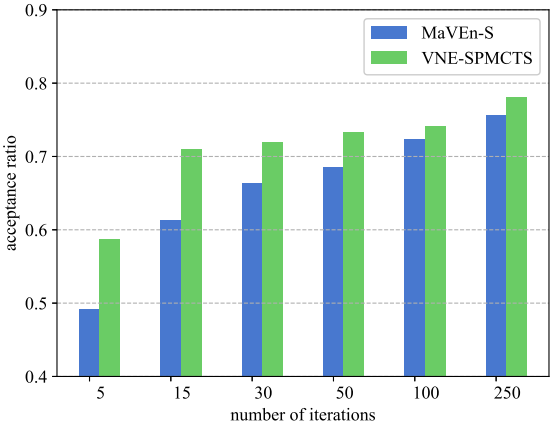

Fig. 12 Acceptance ratio under different numbers of iterations

6.3 Performance in different numbers of iterations

The second experiment tests the performance of the two MCTS based algorithms MaVEn-S and VNE-SPMCTS under the different numbers of iterations, which can be seen as a kind of computational budget. The results are shown in Fig. 11, Fig. 12, and Fig. 13.

The results in Fig. 11, Fig. 12, and Fig. 13 show that the performance of both algorithms increases as the number of iterations increases. However, our proposed VNE-SPMCTS algorithm can increase more quickly when the number of iterations is small. We can observe that the performance of our method in only 15 iterations is comparable to that of MaVEn-S (the basic MCTS) in 100 iterations. The performance of VNE-SPMCTS is also better than that of the MaVEn-S algorithm when the number of iterations is very large (i.e., when the number of iterations is larger than 100). This means that our method can achieve the same performance by using fewer computational resources. It also shows that our added node importance in MCTS can guide the agent to search for a better direction earlier and can speed up the search.

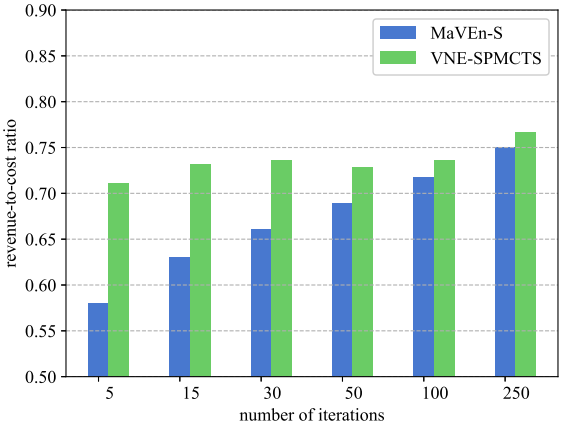

Fig. 13 Long-term revenue-to-cost ratio (R/C) under different numbers of iterations

7 Conclusion

In this paper, we transform the conventional VNE problem into an RL problem by formulating the embedding process as a coordinated two-stage based MDP. Through the reward delay mechanism of the MDP, the link mapping stage can be associated with the node mapping stage. To solve this MDP, we propose a single-player Monte Carlo tree search approach combined with the node importance, where the domain-specific knowledge is used in the Expansion and Simulation stage to speed up the search and more accurately estimate the expected value. Instead of testing which node importance metric performs best, we focus on the problem of how to combine node importance with the basic MCTS. Two ways of utilizing node importance are proposed in our paper: determining the embedding order of virtual nodes before the MCTS begins and determining the probability of substrate nodes being selected during the MCTS. The evaluation results show that our proposed VNE-SPMCTS outperform the typical meta-heuristic algorithm VNE-UEPSO [33] and the MCTS based algorithm MaVEn-S [17] in terms of the average physical node utilization ratio, acceptance ratio, and long-term revenue-to-cost ratio.

References

1. A.K. Bashir, R. Arul, R. Jayaram, A. Arulappan, and S. B. Prathiba. An optimal multitier resource allocation of cloud RAN in 5G using machine learning. *Transactions on Emerging Telecommunications Technologies*, e3627, Aug. 2019.
2. Ali Kashif Bashir, Yuichi Ohsita, and Masayuki Murata. Abstraction layer based distributed architecture for virtualized data centers. 2015.
3. N.M.M.K. Chowdhury and R Boutaba. Network virtualization: state of the art and research challenges. *IEEE Commun. Mag.*, 47(7):20–26, jul 2009.
4. N M Mosharaf Kabir Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. Virtual network embedding with coordinated node and link mapping. In *IEEE INFOCOM 2009*, pages 783–791. IEEE, 2009.
5. Cong Wang, Shashank Shanbhag, and Tilman Wolf. Virtual network mapping with traffic matrices. In *2012 IEEE International Conference on Communications (ICC)*, pages 2717–2722. IEEE, 2012.
6. Haotong Cao, Longxiang Yang, Zeyuan Liu, and Mengting Wu. Exact solutions of vne: A survey. *China communications*, 13(6):48–62, 2016.
7. Haotong Cao, Han Hu, Zhicheng Qu, and Longxiang Yang. Heuristic solutions of virtual network embedding: A survey. *China Communications*, 15(3):186–219, 2018.
8. Peiying Zhang, Haipeng Yao, and Yunjie Liu. Virtual Network Embedding Based on the Degree and Clustering Coefficient Information. *IEEE Access*, 4:8572–8580, 2016.
9. Xiang Cheng, Sen Su, Zhongbao Zhang, Hanchi Wang, Fangchun Yang, Yan Luo, and Jie Wang. Virtual network embedding through topology-aware node ranking. *Comput. Commun. Rev.*, 41(2):39–47, apr 2011.
10. Haotong Cao, Yongxu Zhu, Longxiang Yang, and Gan Zheng. A Efficient Mapping Algorithm with Novel Node-Ranking Approach for Embedding Virtual Networks. *IEEE Access*, 5(1):22054–22066, jan 2017.
11. Peiying Zhang, Haipeng Yao, and Yunjie Liu. Virtual Network Embedding Based on Computing, Network, and Storage Resource Constraints. *IEEE Internet Things J.*, 5(5):3298–3304, oct 2018.
12. Peiying Zhang, Haipeng Yao, Chao Qiu, and Yunjie Liu. Virtual network embedding using node multiple metrics based on simplified ELECTRE method. *IEEE Access*, 6:37314–37327, 2018.
13. Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
14. Ali Kashif Bashir, Yuichi Ohsita, and Masayuki Murata. Abstraction layer based virtual data center architecture for network function chaining. In *2016 IEEE 36th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, 2016.
15. Richard E Korf. Depth-first iterative-deepening. *Artif. Intell.*, 27(1):97–109, sep 1985.
16. Maciej Swiechowski and Jacek Mandziuk. Self-Adaptation of Playing Strategies in General Game Playing. *IEEE Trans. Comput. Intell. AI Games*, 6(4):367–381, dec 2014.
17. Soroush Haeri and Ljiljana Trajković. Virtual network embedding via Monte Carlo tree search. *IEEE Trans. Cybern.*, 48(2):510–521, 2018.
18. Oussama Soualah, Ilhem Fajjari, Nadjib Aitsaadi, and Abdelhamid Mellouk. A batch approach for a survivable virtual network embedding based on monte-carlo tree search. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 36–43. IEEE, 2015.
19. Haipeng Yao, Xu Chen, Maozhen Li, Peiying Zhang, and Luyao Wang. A novel reinforcement learning algorithm for virtual network embedding. *Neurocomputing*, 284:1–9, apr 2018.
20. Maarten P D Schadd, Mark H M Winands, H Jaap Van Den Herik, Guillaume MJ-B Chaslot, and Jos W H M Uiterwijk. Single-player monte-carlo tree search. In *Int. Conf. Comput. Games*, pages 1–12, 2008.
21. Andreas Fischer, Juan Felipe Botero, Michael Till Beck, Hermann De Meer, and Xavier Hesselbach. Virtual network embedding: A survey. *IEEE Commun. Surveys Tuts.*, 15(4):1888–1906, 2013.
22. James Kennedy. Particle swarm optimization. *Encycl. Mach. Learn.*, pages 760–766, 2010.
23. Marco Dorigo and Mauro Birattari. *Ant colony optimization*. Springer, 2010.
24. David E Goldberg. *Genetic algorithms*. Pearson Education India, 2006.
25. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
26. Cong Wang, Yian Su, Lixin Zhou, Sancheng Peng, Ying Yuan, and Hongtao Huang. A virtual network embedding algorithm based on hybrid particle swarm optimization. In *Int. Conf. Smart Comput. Commun.*, pages 568–576, 2016.
27. Rashid Mijumbi, Juan-Luis Gorricho, Joan Serrat, Maxim Claeys, Filip De Turck, and Steven Latré. Design and evaluation of learning algorithms for dynamic resource management in virtual networks. In *2014 IEEE Netw. Oper. Manag. Symp.*, pages 1–9, 2014.
28. Haipeng Yao, Bo Zhang, Peiying Zhang, Sheng Wu, Chunxiao Jiang, and Song Guo. RDAM: A Reinforcement Learning Based Dynamic Attribute Matrix Representation for Virtual Network Embedding. *IEEE Trans. Emerg. Top. Comput.*, 2018.
29. Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. Rethinking virtual network embedding: substrate support for path splitting and migration. *ACM SIGCOMM Comput. Commun. Rev.*, 38(2):17–29, 2008.
30. Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Eur. Conf. Mach. Learn.*, pages 282–293, 2006.
31. David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

32. Robert W Floyd. Algorithm 97: shortest path. *Commun. ACM*, 5(6):345, 1962.
33. Zhongbao Zhang, Xiang Cheng, Sen Su, Yiwen Wang, Kai Shuang, and Yan Luo. A unified enhanced particle swarm optimization-based virtual network embedding algorithm. *Int. J. Commun. Syst.*, 26(8):1054–1073, 2013.
34. Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César A F De Rose, and Rajkumar Buyya. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exp.*, 41(1):23–50, jan 2011.