Virtual Network Embedding via Monte Carlo Tree Search

Article *in* IEEE Transactions on Cybernetics · February 2017

DOI: 10.1109/TCYB.2016.2645123

CITATIONS
57

READS
244

2 authors:

Soroush Haeri
Simon Fraser University
16 PUBLICATIONS **180** CITATIONS
SEE PROFILE

Ljiljana Trajkovic
Simon Fraser University
194 PUBLICATIONS **2,317** CITATIONS
SEE PROFILE

Virtual Network Embedding via Monte Carlo Tree Search

Soroush Haeri, *Student Member, IEEE* and Ljiljana Trajković, *Fellow, IEEE*

***Abstract*—Network virtualization helps overcome shortcomings of the current Internet architecture. The virtualized network architecture enables coexistence of multiple virtual networks on an existing physical infrastructure. Virtual Network Embedding (VNE) problem, which deals with the embedding of virtual network components onto a physical network, is known to be $\mathcal{NP}$-hard.**

In this paper, we propose two VNE algorithms: MaVEn-M and MaVEn-S. MaVEn-M employs the Multi-Commodity Flow algorithm for virtual link mapping while MaVEn-S uses the shortest-path algorithm. They formalize the Virtual Node Mapping problem by using the Markov Decision Process (MDP) framework and devise action policies (node mappings) for the proposed MDP using the Monte Carlo Tree Search algorithm. Service providers may adjust the execution time of the MaVEn algorithms based on the traffic load of virtual network requests. The objective of the algorithms is to maximize the profit of Infrastructure Providers. We develop a discrete event VNE simulator to implement and evaluate performance of MaVEn-M, MaVEn-S, and several recently proposed VNE algorithms. We introduce *profitability* as a new performance metric that captures both acceptance and revenue to cost ratios. Simulation results show that the proposed algorithms find more profitable solutions than the existing algorithms. Given additional computation time, they further improve embedding solutions.

I. Introduction

The best-effort service, supported by the current Internet architecture, is not well-suited for all applications. A significant barrier to innovation has been imposed by the inability of the current Internet architecture to support a diverse array of applications [1]. The great success of the Internet has increased its ubiquity that, consequently, has lead to various challenges that the current Internet architecture is unable to address [2]. Network virtualization overcomes these shortcomings [1], [3]. The virtualized network model divides the role of Internet Service Providers (ISPs) into two independent entities: Infrastructure Providers (InPs) and Service Providers (SPs). The InPs manage the physical infrastructure while the SPs aggregate resources from multiple InPs into multiple Virtual Networks (VNs) to provide end-to-end services [3], [4].

In the virtualized network architecture, an InP owns and operates a *substrate network* composed of physical nodes and links that are interconnected in an arbitrary topology. Combinations of the substrate network nodes and links are used to embed various virtualized networks. Virtual networks embedding enables end-to-end service provisioning without requiring unified protocols, applications, or control and management planes [5].

The authors are with the School of Engineering Science, Simon Fraser University, Vancouver, BC V5A 1S6, Canada (e-mail: shaeri@sfu.ca; ljilja@sfu.ca).

An InP's revenue depends on the resource utilization within the substrate network that, in turn, depends on the efficiency of the algorithm that allocates the substrate network resources to virtual networks [4]. This resource allocation is known as the virtual network embedding (VNE) [6], which may be formulated as a mixed-integer program (MIP) [7] or may be reduced to the multiway separator problem [8], [9]. Both problems are $\mathcal{NP}$-hard making the VNE problem also $\mathcal{NP}$-hard. This is one of the main challenges in network virtualization.

MIPs have already been employed to solve the VNE problem [7], [10], [11]. The proposed R-ViNE and D-ViNE algorithms [7] use a rounding-based approach to attain a linear programming relaxation of the MIP that corresponds to the VNE problem [7]. Their objective is to minimize the cost of accommodating Virtual Network Requests (VNRs). Node-ranking-based algorithms are among the most recent approaches to solve the VNE [12]–[14]. This family of algorithms computes a score/rank for substrate and virtual nodes based on various heuristics. Then, using the computed ranks, a *large-to-large and small-to-small* [12] mapping scheme is employed to map the virtual nodes onto substrate nodes. The Global Resource Capacity (GRC) [14] is among the most recent node-ranking-based algorithms that outperforms the earlier similar algorithms. Subgraph isomorphism detection [15], particle swarm optimization [16], and ant colony optimization [17] are among other employed methods.

The VNE problem may be divided into two subproblems: Virtual Node Mapping (VNoM) and Virtual Link Mapping (VLiM). VNoM algorithms map virtual nodes onto substrate nodes while VLiM algorithms map virtual links onto substrate paths. Algorithms that have been proposed for solving VNE are categorized into three categories depending on the approaches taken to solve these two subproblems [18]. The *uncoordinated two-stage algorithms* first solve the VNoM problem and provide a node mapping to the VLiM solver. In these approaches, the VNoM and VLiM solvers operate independently without any coordination [9], [19]. The *coordinated two-stage algorithms* also first solve the VNoM problem. Unlike the uncoordinated solutions, these algorithms consider the virtual link mappings when solving the VNoM problem [7], [12], [14]. The *coordinated one-stage algorithms* solve the VNoM and VLiM problems simultaneously. When two virtual nodes are mapped, also mapped is the virtual link connecting the two nodes [20].

Most VNE algorithms proposed in the literature solve the VNoM problem while using the shortest-path algorithms (k-shortest path, Breadth-First Search (BFS), and Dijkstra) or the Multi-Commodity Flow (MCF) algorithm to solve VLiM.

Unlike the MCF algorithm, the shortest-path algorithms do not allow path splitting. Path splitting [9] enables a virtual link to be mapped onto multiple substrate paths.

Our contributions are: modeling VNoM as a Markov Decision Process (MDP), introducing two Monte Carlo Tree Search-based algorithms, and developing a VNE simulator.

We model the VNoM problem as an MDP. MDPs decompose sequential decision-making problems into states, actions, transition probabilities between the states given the actions, and the received rewards for performing actions in given states.

We introduce two *M*onte C*a*rlo Tree Search-based *V*irtual Network *E*mbedd*i*ng algorithms: MaVEn-M and MaVEn-S. They are coordinated two-stage VNE algorithms that solve the proposed MDP for VNoM using the Monte Carlo Tree Search (MCTS) algorithm [21], [22]. MaVEn-M employs the MCF algorithm to coordinate VNoM and VLiM when solving the VNoM subproblem. It also employs MCF to solve the VLiM subproblem after it obtains the VNoM solution. MaVEn-S employs a simple BFS algorithm. A number of existing VNE algorithms find only one solution for virtual network mapping and they are unable to improve the solution even if additional execution time is available [7], [12]–[14]. One advantage of the proposed algorithms is that their runtime may be adjusted according to the VNR arrival rates. If the VNR arrival rate is low, their execution time may be increased to find more profitable embedding solutions.

We develop a VNE simulator *VNE-Sim* written in C++. It is based on the *Discrete Event System Specification* (DEVS) framework [23] and employs the *Adevs* library [24]. For performance evaluation, we implement the proposed MaVEn-M and MaVEn-S algorithms, the well-known MIP-based R-ViNE and D-ViNE algorithms [7], and the recently proposed node-ranking-based GRC algorithm [14]. We also introduce *profitability* as a new metric for comparing VNE algorithms.

The remainder of this manuscript is organized as follows. In Section II, we present the VNE problem and its objective function, establish its upper bound, and introduce profitability as a performance metric. An MDP formulation of the VNoM problem is proposed in Section III. We then introduce two MaVEn algorithms, which utilize MCTS for finding optimal action policies for the proposed MDP. In Section IV, performance of the MaVEn algorithms is compared to the existing VNE algorithms. We conclude with Section V.

II. Virtual Network Embedding Problem

Let $G^s(N^s, E^s)$ denote the substrate network graph, where $N^s = \{n_1^s, n_2^s, \ldots, n_j^s\}$ is the set of j substrate nodes (vertices) while $E^s = \{e_1^s, e_2^s, \ldots, e_k^s\}$ is the set of k substrate edges (links). Let the i^{th} VNR be denoted by a triplet $\Psi_i(G^{\Psi_i}, \omega^{\Psi_i}, \xi^{\Psi_i})$, where $G^{\Psi_i}(N^{\Psi_i}, E^{\Psi_i})$ is the virtual network graph with $N^{\Psi_i} = \{n_1^{\Psi_i}, n_2^{\Psi_i}, \ldots, n_\ell^{\Psi_i}\}$ and $E^{\Psi_i} = \{e_1^{\Psi_i}, e_2^{\Psi_i}, \ldots, e_m^{\Psi_i}\}$ denoting the set of ℓ virtual nodes and m virtual edges, respectively. Furthermore, ω^{Ψ_i} is the VNR arrival time and ξ^{Ψ_i} is its life-time according to distributions Ω and Ξ, respectively.

Substrate nodes and edges possess resources such as residual CPU capacity and bandwidth that may be used for embedding virtual network elements. VNoM algorithms assign a virtual node n^{Ψ_i} to a substrate node n^s that satisfies requirements of the virtual node. We denote such mapping by a tuple (n^{Ψ_i}, n^s). VLiM algorithms establish a virtual link using one or more substrate links. If a virtual link e^{Ψ_i} is established using q substrate links $\{e_1^s, e_2^s, \ldots, e_q^s\}$, we denote such mapping by a tuple $\big(e^{\Psi_i}, \{e_1^s, e_2^s, \ldots, e_q^s\}\big)$. The goal of VNoM and VLiM algorithms is to optimize an objective function.

We assume that substrate nodes possess residual CPU capacities $\mathcal{C}(n^s)$ and are located at coordinates $\mathcal{L}(n^s) = (x_{n^s}, y_{n^s})$. Virtual nodes require a CPU capacity $\mathcal{C}(n^{\Psi_i})$ and have a location preference $\mathcal{L}(n^{\Psi_i}) = (x_{\Psi_i}, y_{\Psi_i})$. The only assumed substrate link resource is the residual link bandwidth $\mathcal{B}(e^s)$. Virtual links have bandwidth requirements $\mathcal{B}(e^{\Psi_i})$ [7]. Assuming that path splitting [9] is permitted for link mapping, a substrate node is eligible to host a virtual node if:

$$\mathcal{C}(n^s) \geq \mathcal{C}(n^{\Psi_i}), \tag{1}$$

$$d\big(\mathcal{L}(n^s), \mathcal{L}(n^{\Psi_i})\big) \leq \delta^{\Psi_i}, \tag{2}$$

$$\sum_{e^s \in E_{n^s}} \mathcal{B}(e^s) \geq \sum_{e^{\Psi_i} \in E_{n^{\Psi_i}}} \mathcal{B}(e^{\Psi_i}). \tag{3}$$

where $d(.,.)$ is the Euclidean distance function, δ^{Ψ_i} is a predetermined maximum allowable distance for the node embeddings of Ψ_i, E_{n^s} is the set of substrate links connected to a substrate node n^s, and $E_{n^{\Psi_i}}$ is the set of virtual links connected to the virtual node n^{Ψ_i}.

If path splitting is not permitted, (3) is not a sufficient condition, and a substrate node is eligible for embedding a virtual node if there exist at least one mapping $\mathcal{U} = \{(e_m^{\Psi_i}, e_k^s), \ldots, (e_w^{\Psi_i}, e_v^s)\}$ from $E_{n^{\Psi_i}}$ to E_{n^s} such that:

$$\forall\, (e_m^{\Psi_i}, e_n^s) \in \mathcal{U}, \sum_{e^{\Psi_i} \in \mathcal{U}(e_n^s)} \mathcal{B}(e^{\Psi_i}) \leq \mathcal{B}(e_n^s), \tag{4}$$

where $\mathcal{U}(e_n^s)$ denotes the set of all virtual links e^{Ψ_i} that are mapped to the substrate link e_n^s by the mapping $\mathcal{U}$. The substrate nodes that satisfy these conditions form the set of candidate nodes $N^s(n^{\Psi_i})$ for embedding n^{Ψ_i}.

A. Objective of Virtual Network Embedding

Majority of the proposed VNE algorithms have the objective to maximize the profit of InPs [7], [12]–[14]. Embedding revenue, cost, and the VNR acceptance ratio are the three main contributing factors to the generated profit.

Revenue: InPs generate revenue by embedding VNRs. The revenue generated by embedding a VNR Ψ_i is calculated as a weighted sum of VNR resource requirements:

$$\mathbf{R}(G^{\Psi_i}) = w_c \sum_{n^{\Psi_i} \in N^{\Psi_i}} \mathcal{C}(n^{\Psi_i}) + w_b \sum_{e^{\Psi_i} \in E^{\Psi_i}} \mathcal{B}(e^{\Psi_i}), \tag{5}$$

where w_c and w_b are the weights for CPU and bandwidth requirements, respectively [13]. The network provider receives a revenue only if the virtual network request is accepted for embedding. No revenue is generated in the case the request is rejected. In this paper, we assume that requests are served one at a time [13], [14]. Furthermore, we are not considering future

VNR arrivals. Hence, to maximize the revenue, the service provider should try to accept as many requests as possible. The generated revenue only depends on whether or not the request is accepted. If a request is accepted for embedding, the generated revenue (5) does not depend on the substrate resources used to serve the request.

Cost: For embedding a VNR Ψ_i, the InP incurs a cost based on the resources it allocates for embedding the VNR. The incurred cost is calculated as:

$$\mathbf{C}(G^{\Psi_i}) = \sum_{n^{\Psi_i} \in N^{\Psi_i}} \mathcal{C}(n^{\Psi_i}) + \sum_{e^{\Psi_i} \in E^{\Psi_i}} \sum_{e^s \in E^s} f_{e^s}^{e^{\Psi_i}}, \quad (6)$$

where $f_{e^s}^{e^{\Psi_i}}$ denotes the total bandwidth of the substrate edge e^s that is allocated for the virtual edge e^{Ψ_i} [7], [9]. Unlike the revenue function (5), the cost (6) depends on the embedding configuration. Hence, if a VNR Ψ_i is accepted, the cost $\mathbf{C}(G^{\Psi_i})$ values depend on the embedding configuration within the substrate network.

Acceptance Ratio: In a given time interval τ, the ratio of the number of accepted VNRs $|\Psi^a(\tau)|$ to the total number of VNRs that arrived $|\Psi(\tau)|$ defines the acceptance ratio or the probability of accepting a VNR:

$$p_a^\tau = \frac{|\Psi^a(\tau)|}{|\Psi(\tau)|}. \quad (7)$$

Objective Function: Similar to other proposed algorithms [7], [12], [14], we also aim to maximize the InP profit defined as the difference between the generated revenue and cost. Maximizing the revenue and acceptance ratio while minimizing the cost of VNR embeddings maximizes the generated profit of InPs. Therefore, we define the objective of embedding a VNR Ψ_i as maximizing the following objective function:

$$\mathcal{F}(\Psi_i) = \begin{cases} \mathbf{R}(G^{\Psi_i}) - \mathbf{C}(G^{\Psi_i}) & \text{successful embeddings} \\ \Gamma & \text{otherwise} \end{cases}, \quad (8)$$

where Γ defines the greediness of the embedding. Assuming that $\mathbf{R}(G^{\Psi_i}) - \mathbf{C}(G^{\Psi_i}) \in [a, b]$, setting Γ to a value smaller than a results in a greedy VNR embedding because in this case Γ is the lower bound of $\mathcal{F}$. Hence, to maximize $\mathcal{F}$ (8), any successful embedding is better than rejecting the VNR. Assigning $\Gamma \in [a, b]$ introduces a preferential VNR acceptance and, thus, if

$$\mathbf{R}(G^{\Psi_i}) - \mathbf{C}(G^{\Psi_i}) < \Gamma, \quad (9)$$

rejecting the VNR Ψ_i maximizes $\mathcal{F}(\Psi_i)$. We only consider the greedy approach by assigning a large negative penalty for unsuccessful embeddings ($\Gamma \to -\infty$).

In order to define the upper bound of the objective function, let us consider the case where path splitting is not permitted. In this case, (6) becomes:

$$\mathbf{C}(G^{\Psi_i}) = \sum_{n^{\Psi_i} \in N^{\Psi_i}} \mathcal{C}(n^{\Psi_i}) + \sum_{e^{\Psi_i} \in E^{\Psi_i}} \eta_{e^{\Psi_i}} \mathcal{B}(e^{\Psi_i}), \quad (10)$$

where $\eta_{e^{\Psi_i}}$ denotes the length of the substrate path used to accommodate the virtual edge e^{Ψ_i}. Hence:

$$\begin{aligned} &\mathbf{R}(G^{\Psi_i}) - \mathbf{C}(G^{\Psi_i}) \\ &\quad = w_c \sum_{n^{\Psi_i} \in N^{\Psi_i}} \mathcal{C}(n^{\Psi_i}) - \sum_{n^{\Psi_i} \in N^{\Psi_i}} \mathcal{C}(n^{\Psi_i}) \\ &\quad + w_b \sum_{e^{\Psi_i} \in E^{\Psi_i}} \mathcal{B}(e^{\Psi_i}) - \sum_{e^{\Psi_i} \in E^{\Psi_i}} \eta_{e^{\Psi_i}} \mathcal{B}(e^{\Psi_i}) \\ &\quad = (w_c - 1) \sum_{n^{\Psi_i} \in N^{\Psi_i}} \mathcal{C}(n^{\Psi_i}) \\ &\quad + \sum_{e^{\Psi_i} \in E^{\Psi_i}} (w_b - \eta_{e^{\Psi_i}}) \mathcal{B}(e^{\Psi_i}). \end{aligned} \quad (11)$$

The substrate path lengths $\eta_{e^{\Psi_i}} \forall e^{\Psi_i} \in E^{\Psi_i}$ are the only parameters that depend on the embedding configuration. Therefore, (11) is maximized when the path lengths are minimized. The minimum substrate path length for embedding a virtual link is equal to 1. Hence:

$$\begin{aligned} &\max\{\mathbf{R}(G^{\Psi_i}) - \mathbf{C}(G^{\Psi_i})\} \\ &\quad = (w_c - 1) \sum_{n^{\Psi_i} \in n^{\Psi_i}} \mathcal{C}(n^{\Psi_i}) \\ &\quad + (w_b - 1) \sum_{e^{\Psi_i} \in E^{\Psi_i}} \mathcal{B}(e^{\Psi_i}). \end{aligned} \quad (12)$$

In order to remove the influence of the weights on the calculations of the upper bound, without loss of generality we may assume $w_c = w_b = 1$ [7], [9], [12]. Hence:

$$\max\{\mathbf{R}(G^{\Psi_i}) - \mathbf{C}(G^{\Psi_i})\} = 0. \quad (13)$$

The upper bound of the objective function is achieved when Ψ_i is successfully embedded (8). Therefore, we define the upper bound for the objective function as:

$$\begin{aligned} \max\{\mathcal{F}(\Psi_i)\} &= \max\{\mathbf{R}(G^{\Psi_i}) - \mathbf{C}(G^{\Psi_i})\} \\ &\triangleq \mathcal{F}^{ub}(\Psi_i). \end{aligned} \quad (14)$$

The same upper bound would be achieved if path splitting was permitted because the minimum substrate path length for embedding a virtual link might not be less than 1.

B. VNE Performance Metrics

Acceptance ratio, revenue to cost ratio, and substrate network resource utilization are the main VNE performance metrics [7], [13], [14]. Considering acceptance and revenue to cost ratios independently does not adequately estimate performance of VNE algorithms. For example, high acceptance ratio when the average revenue to cost ratio is low is undesirable because it leaves the substrate resources underutilized [7]. The same applies to having a high revenue to cost ratio while having a low acceptance ratio. Therefore, acceptance and average revenue to cost ratios should be considered simultaneously. Hence, we introduce profitability θ as a new performance measure. The profitability θ in a time interval τ is calculated as a product of acceptance and revenue to cost ratios:

$$\theta = p_a^\tau \times \frac{\sum_{\Psi_i \in \Psi^a(\tau)} \mathbf{R}(G^{\Psi_i})}{\sum_{\Psi_i \in \Psi^a(\tau)} \mathbf{C}(G^{\Psi_i})}, \quad (15)$$

where p_a^τ is the acceptance ratio during the interval τ (7) and $\Psi^a(\tau)$ is the set of all accepted VNRs in the interval τ. Since the CPU and bandwidth revenue weights are $w_c = w_b = 1$, the maximum profitability $\theta_{max} = 1$. Higher profitability implies that the algorithm has high acceptance and high revenue to cost ratios. Therefore, VNE algorithms are desired to have profitability values close to 1.

III. Virtual Network Embedding as a Markov Decision Process

We model the VNE problem as a sequence of decision-making instances where an agent (VNE algorithm) receives VNRs. The agent solves the VNoM and VLiM for each VNR Ψ_i and receives a reward $\mathcal{F}(\Psi_i)$. The agent's objective is to maximize this reward. (Throughout this paper, we interchangeably use terms agent and decision-making agent.)

A. Markov Decision Process

We use Markov Decision Process (MDP) to model the sequential decision-making problem. A discrete time MDP $\mathcal{M}$ is a quintuple $(\mathcal{T}, \Phi, \mathcal{A}, R, P)$, where $\mathcal{T}$ is the set of decision-making instances, Φ is the state space, $\mathcal{A}$ is the action space, $R : \Phi \times \mathcal{A} \to \mathbb{R}$ is the reward function that assigns real-valued rewards to state-action pairs, and $P : \Phi \times \mathcal{A} \times \Phi \to [0, 1]$ is a transition probability distribution. Therefore, $R(\phi_t = \phi, a_t = a)$ is the reward for performing action a in state ϕ and

$$P(\phi', a, \phi) = \Pr(\phi_{t+1} = \phi' | a_t = a, \phi_t = \phi) \sim P \qquad (16)$$

is the probability of a transition to state ϕ' when selecting action a in state ϕ. A state ρ is called a *terminal* state if $P(\rho, a, \rho) = 1$. We denote the reward of entering a terminal state by R_ρ. An MDP $\mathcal{M}$ is called *episodic* if it possesses a terminal state [25].

The behavior of a decision-making agent is defined by its policy π for selecting actions. The overall return of a given policy π is calculated as:

$$\mathcal{R}^\pi = R_\rho^\pi + \sum_{t=1}^{T} \gamma^t R_t^\pi, \qquad (17)$$

where T defines the decision-making *horizon* and $0 \leq \gamma^t \leq 1$ is a *discount factor*. The MDP is called *finite-horizon* if $T < \infty$ while $T \to \infty$ defines an *infinite-horizon* MDP. The goal of the decision-making agent is to find policy π^* that maximizes the expected cumulative rewards given an initial state ϕ_1 [26].

B. A Finite-Horizon MDP Model for Coordinated VNoM

Let us consider a substrate network with j nodes and k links. At an arbitrary time instant ω^{Ψ_i}, the VNE solver receives a VNR Ψ_i that requires ℓ virtual nodes and m virtual links. We define the MDP corresponding to virtual node mapping of Ψ_i as a finite-horizon MDP $\mathcal{M}^{\Psi_i}$. The decision-making agent consecutively selects ℓ substrate nodes for embedding nodes of Ψ_i, yielding to ℓ decision-making instances at discrete times t until the horizon $T = \ell$ is reached. If all nodes are successfully mapped, the process reaches its terminal state at $t = \ell + 1$. We assume that in a given state $\phi_q^{\Psi_i}$, the agent tries to identify a substrate node $n^s \in N_q^s$ for embedding the *first element* $n_q^{\Psi_i}$ of the set $N_q^{\Psi_i}$, where N_q^s the set of all substrate nodes that are available for embedding virtual nodes at state $\phi_q^{\Psi_i}$ and $N_q^{\Psi_i}$ is the ordered set of all virtual nodes that are yet to be embedded.

The state of $\mathcal{M}^{\Psi_i}$ at the decision making instance t is defined as:

$$\phi_t^{\Psi_i} = \left(N_t^{\Psi_i} = N_{t-1}^{\Psi_i} \setminus \{n_{t-1}^{\Psi_i}\}, N_t^s = N_{t-1}^s \setminus \{n_{t-1}^s\}\right), \qquad (18)$$

where n_{t-1}^s is the substrate node selected for embedding the virtual node $n_{t-1}^{\Psi_i}$ in the previous time step. In the initial state, no virtual node has been embedded and, thus, all substrate nodes are available for embedding the first virtual node. Hence, $N_1^{\Psi_i} = N^{\Psi_i}$ and $N_1^s = N^s$.

The agent selects a node $n^s \in \{N_t^s \cap N^s(n_t^{\Psi_i})\}$ from the set of viable actions:

$$\mathcal{A}_t^{\Psi_i} = \{\varepsilon\} \cup \left\{(n_t^{\Psi_i}, n^s) : \forall n^s \in \{N_t^s \cap N^s(n_t^{\Psi_i})\}\right\}, \qquad (19)$$

where ε denotes an arbitrary action that forces the transition to a terminal state. As the result of selecting a substrate node n_t^s for embedding the virtual node $n_t^{\Psi_i}$, the agent receives a reward and $\mathcal{M}^{\Psi_i}$ transits to state $\phi_{t+1}^{\Psi_i}$.

The state transition (18) occurs because multiple virtual nodes cannot be embedded into a single substrate node. Depending on the choice of the substrate node for embedding the virtual node $n_t^{\Psi_i}$ at decision making instance t, there are $|\mathcal{A}_t^{\Psi_i}|$ possible next states, where $|\mathcal{A}_t^{\Psi_i}|$ denotes the number of viable actions in state $\phi_t^{\Psi_i}$. The probability of the state transition is:

$$\Pr(\phi_{t+1}^{\Psi_i} \mid n_t^s, \phi_t^{\Psi_i}) = 1. \qquad (20)$$

The process continues until $\mathcal{M}^{\Psi_i}$ transits to the ℓ^{th} state $\phi_\ell^{\Psi_i}$ where an action from $\mathcal{A}_\ell^{\Psi_i}$ is selected. The decision-making horizon is then reached and $\mathcal{M}^{\Psi_i}$ transits to a terminal state where the reward R_ρ is calculated. We assume that the immediate rewards R_t for all $t \leq \ell$ are zero and the agent receives a reward R_ρ only when it reaches a terminal state because partially mapping a virtual network does not necessarily lead to a successful complete mapping. Reaching a terminal state when $t < \ell$ implies that there is no eligible substrate node for mapping a virtual node n^{Ψ_i}. Hence, $N^s(n^{\Psi_i}) = \varnothing$). Therefore, ε that forces $\mathcal{M}^{\Psi_i}$ to its terminal state has been selected. This implies that the VNoM has been unsuccessful thus $R_\rho = \Gamma$ (8). On the contrary, reaching a terminal state when $t = \ell + 1$ implies that the VNoM has been successful. The agent then proceeds to solve VLiM. If VLiM is successful, the VNR is accepted for embedding and the agent receives the reward $R_\rho = \mathbf{R}(G^{\Psi_i}) - \mathbf{C}(G^{\Psi_i})$. Otherwise, the VNR is rejected and $R_\rho = \Gamma$ (8).

We do not consider link mappings at the intermediate states of $\mathcal{M}^{\Psi_i}$. However, the agent is unable to select an optimal action policy π^* without knowing the final reward R_ρ, which requires solving the VLiM problem. Hence, finding π^* for $\mathcal{M}^{\Psi_i}$ results in a coordinated VNoM solution [18].

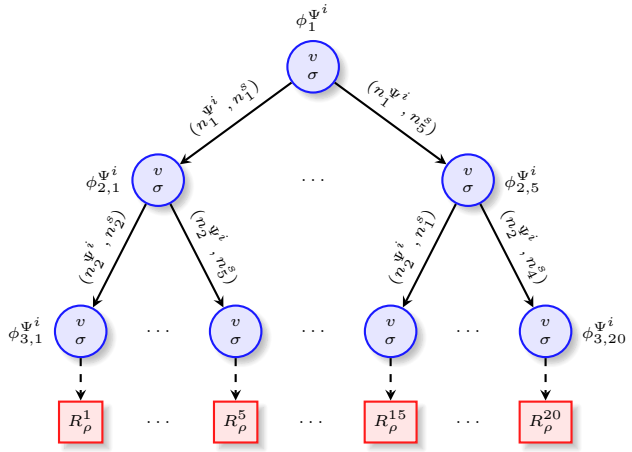

Fig. 1. Example of a VNoM search tree for embedding a VNR Ψ^i with 3 nodes onto a substrate network with 5 nodes.

C. Monte Carlo Tree Search for Solving the VNoM MDP

The size of the MDP state space grows exponentially with the number of state variables. Complexity of exact algorithms for solving MDP such as the *policy iteration* and the *value iteration* [26] is polynomial in the size of the state space [27]. Therefore, finding exact solutions for MDPs with large number of state variables is intractable. Solving such MDPs often involves finding a *near-optimal* solution. The number of state variables of the proposed MDP $\mathcal{M}^{\Psi_i}$ depends on the number of substrate and virtual nodes $|N^s|$ and $|N^{\Psi_i}|$, respectively. Consequently, the number of $\mathcal{M}^{\Psi_i}$ states grows exponentially with $|N^s|$ and $|N^{\Psi_i}|$. Hence, finding an exact solution for $\mathcal{M}^{\Psi_i}$ is intractable for even a fairly small $|N^s|$ and $|N^{\Psi_i}|$.

Various approaches have been proposed for finding near-optimal solutions of large MDPs [28]–[31]. Recent approaches [30], [31] are based on the Monte Carlo Tree Search (MCTS) [21]. They assume that the decision-making agent has access to a generative model $\mathcal{G}$ of the MDP. The model is capable of generating samples of successor states and rewards given a state and an action [31]. The agent uses the model $\mathcal{G}$ to perform a sampling-based look-ahead search [29]. MCTS builds a sparse search tree [32] and selects actions using Monte Carlo samplings. These actions are used to deepen the tree in the most promising direction [30].

Consider the search tree for solving $\mathcal{M}^{\Psi_i}$. Its depth is equal to the horizon T of $\mathcal{M}^{\Psi_i}$. The nodes and edges of the tree correspond to states and actions, respectively. The root of the tree corresponds to the initial state $\phi_1^{\Psi_i}$ of $\mathcal{M}^{\Psi_i}$. Let $|\mathcal{A}_m^{\Psi_i}|$ be the number of available actions at a given state $\phi_m^{\Psi_i}$. The search tree node that corresponds to the state $\phi_m^{\Psi_i}$ has $|\mathcal{A}_m^{\Psi_i}|$ child nodes, each corresponding to a possible next state $\phi_{m+1}^{\Psi_i}$ that is a result of selecting an action $(n_m^{\Psi_i}, n^s) \in \mathcal{A}_m^{\Psi_i}$. Tree node store *values* and *visit counts*. The value of a non-terminal tree node is the cumulative sum of the rewards received in the discovered and reachable terminal nodes. A path from the root to a leaf node defines an action policy π. An example of a VNoM search tree for embedding a VNR with 3 nodes onto a substrate network with 5 nodes is shown in Fig. 1.

MCTS algorithm begins with a tree that only consists of the root node. It then executes four phases until a predefined computational budget β is exhausted:

1) Selection: The tree is traversed from the root until a non-terminal leaf node is reached. At each level of the tree, a child node is selected based on a *selection strategy*. This strategy may explore the undiscovered sections of the search tree to find better actions or may exploit promising subtrees that have already been discovered. This is known as balancing *exploration vs. exploitation* [33].

Various selection strategies have been proposed in the literature [21], [34], [35]. The Upper Confidence Bounds for Trees (UCT) [21] is one of the most commonly used selection strategies. Let u denote the current node of the search tree and $\mathcal{I}$ the set of all its children. Furthermore, let v_i and σ_i denote the value and visit count of a node i, respectively. UCT selects a child κ from:

$$\kappa \in \arg\max_{i \in \mathcal{I}} \left(\frac{v_i}{\sigma_i} + D\sqrt{\frac{\ln \sigma_u}{\sigma_i}}\right), \tag{21}$$

where D is an exploration constant that determines the balance between exploration and exploitation. If $D = 0$, the selection strategy is strictly exploitative.

2) Expansion: After a non-terminal leaf node is selected, one or more of its successors are added to the tree. The most common *expansion strategy* is to add one node for every execution of the four MCTS phases. The new node corresponds to the next state [22].

3) Simulation: From the given state of the non-terminal node that has been selected, a sequence of actions is performed until a terminal state is reached. Even though MCTS converges with randomly selected actions [21], utilizing domain knowledge may improve the convergence speed [30].

4) Backpropagation: The reward is calculated after a terminal state is reached in the *Simulation* phase. This reward is then propagated from the terminal node to the root. Every tree node in the current trajectory is updated by adding the reward to its current value v and incrementing its count σ.

The computational budget β is defined as the number of evaluated action samples per selection cycle. After repeating the four phases β 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 selected child is then chosen to be the new root of the search tree. The process repeats until the root of the search tree is terminal regardless of parameter β.

D. Parallel Monte Carlo Tree Search

Various techniques for parallelizing the MCTS algorithm are available [36]. We consider a symmetric multiprocessor (SMP) system as the platform for parallelization where memory is shared and, hence, mutual exclusions (mutex) should be employed to avoid corruption of the search tree when multiple threads attempt to access and modify the search tree during the phase 1, 2, and 4 of the MCTS algorithm. The Simulation phase does not require information from the search tree thus enabling simulations to be executed independently without any mutex [36]. Root and leaf parallelizations [37] are the common techniques that do not require any mutex. Hence, they are simple to implement and may be executed on

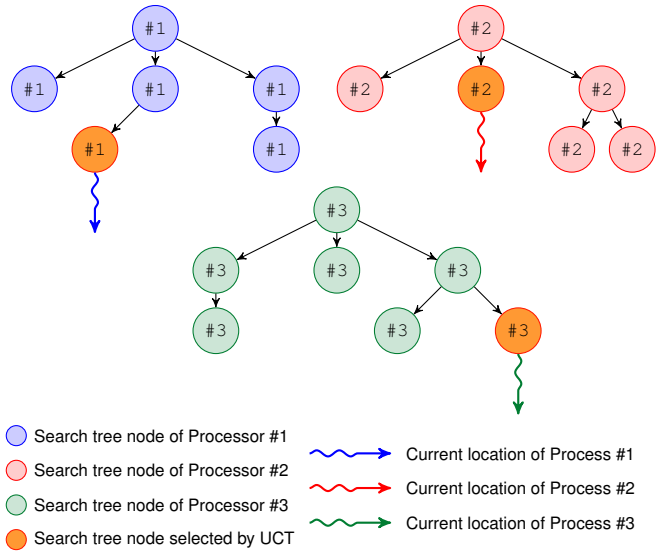

Fig. 2. Root parallelization of Monte Carlo Tree Search. UCT is the Upper Confidence Bound for Trees selection strategy.

distributed memory architectures such as clusters. An example of the root parallelization technique is shown in Fig. 2. It requires less coordination and communication between the processors compared to the leaf parallelization. Furthermore, performance evaluations have shown that root parallelization leads to superior results [36], [37].

Each processor creates its own search tree and the processors do not share information. A unique random number generator seed should be assigned to each processor to ensure that the constructed search trees are not identical. At the end of simulations time (β), each processor communicates the value v and count σ of the children of its root to the master processor, which then calculates the best action based on the information it receives from other processors [37].

E. MaVEn Algorithms

We propose MaVEn-M and MaVEn-S algorithms that employ MCTS to solve the MDP $\mathcal{M}^{\Psi_i}$. Their pseudocode is listed in Algorithm 1. MaVEn-M uses MCF to solve VLiM and to calculate values of terminal states while MaVEn-S uses a breadth first search shortest path algorithm.

Two implementation details should be considered when implementing Algorithm 1: (They have been omitted from the pseudocode due to space constrains.)

Node Creation: When creating the root or new nodes (lines 3 and 46), the child array of the node should be initialized. It is important to note that these child nodes are not yet part of the tree. Therefore, although they have value v and visit count σ, they should be distinguishable from the nodes that are part of the tree (line 45). This is achieved by setting v and σ while not setting their states.

Child Initialization: When initializing the child array of a node, value v and visit count σ of these child nodes should be set. In this stage, we preclude actions that are not viable in the state that corresponds to the parent node. We assume that the parent node corresponds to a state $(N_x^{\Psi_i}, N_x^s)$. In this state, the goal is to find a substrate node $n^s \in N_x^s$ for embedding the first element of $N_x^{\Psi_i}$ denoted by $n_{current}^{\Psi_i}$. Since the viable actions are $\{n^s : \forall n^s \in N^s(n_{current}^{\Psi_i})\}$, we set $v = 0$ and $\sigma = 0$ for the child nodes that correspond to these substrate nodes while setting the v and σ of all other child nodes to large negative and positive values, respectively. This ensures that only substrate nodes in $N^s(n_{current}^{\Psi_i})$ are considered as viable choices for embedding. Hence, the UCT strategy (21) will not select actions that are not viable.

F. Parallelization of the MaVEn Algorithms

MCTS is highly parallelizable [31] and parallelization improves its performance. There are various techniques to successfully parallelize the MCTS algorithm and improve the tree search execution time [36]. We parallelize MaVEn using root parallelization because it is one of the most successful approaches [36], [37]. Let us assume that there are p available processors for parallelization. Each processor is assigned a unique integer number $rank \in \{0, \ldots, p\}$. We assume that the processor with $rank = 0$ is the master responsible for collecting the information from other processors and selecting the best action. The pseudocode of the parallelized MaVEn is shown in Algorithm 2.

IV. Performance Evaluation

In this Section, we present simulation results used to compare the proposed and the existing VNE algorithms. The performance of MCTS is mostly influenced by the computational budget β. Therefore, we compare the algorithms by varying β from 5 to 250 samples per virtual node embedding while keeping the VNR traffic load constant. The remaining MCTS parameters were adopted from the literature [35]. We then compare the algorithms by increasing the VNR traffic load. The MaVEn-M and MaVEn-S simulation scenarios that are presented in Section IV-B and Section IV-C are repeated six times with different seeds for the random number generator. The results are averaged over these executions. Simulations were performed on a Dell Optiplex-790 with 16 GB memory and the Intel Core i7 2600 CPU.

A. Simulation Environment

We developed a discrete event simulator *VNE-Sim* based on the DEVS framework [23] in order to evaluate performance of the proposed MaVEn-M and MaVEn-S algorithms. VNE-Sim is written in C++ and provides the base classes and operations needed to simulate VNE algorithms. It is publicly available to the research community [38]. We implement MaVEn-M, MaVEn-S, D-ViNE [7], R-ViNE [7], and GRC [14] algorithms and compare their acceptance ratio (7), revenue to cost ratio, profitability (15), and average execution time per VNR embedding. In simulations, the exploration constant is set to $D = 0.5$ [35]. Parameters for ViNE and GRC algorithms are adopted from [7] and [14], respectively. The implemented GRC algorithm is modified to consider the preference criteria of the virtual node location (2).

Algorithm 1 Pseudocode of MaVEn algorithm: MaVEn-M employs the MCF algorithm while MaVEn-S uses a breadth first search-based shortest path algorithm to solve VLiM (lines 21, 65, and 88). The *head* keyword (lines 57, 79) refers to the first element of the set N^{Ψ_i}.

1: **procedure** MAVEN($\Psi_i, G^s(N^s, E^s), \beta$)
2: $\phi \leftarrow (N^{\Psi^i}, N^s)$
3: Create $Root$ $(v = 0, \sigma = 0, State = \phi)$
4: $nodesMap[] \leftarrow \varnothing$
5: $vnI \leftarrow 1$
6: **do**
7: $snI \leftarrow$ MCTS $(Root, \beta)$
8: **if** $snI \neq \varepsilon$ **then**
9: $nodesMap.Add\ (n^{\Psi_i}_{vnI}, n^s_{snI})$
10: **if** $Root.child[snI].State$ is terminal **then**
11: $terminate \leftarrow true$
12: **else**
13: $vnI \leftarrow vnI + 1$
14: $Root \leftarrow Root.child[snI]$
15: **end if**
16: **else**
17: $terminate \leftarrow$ true
18: **end if**
19: **while** $terminate \neq$ true
20: **if** $nodesMap.Size = |N^{\Psi^i}|$ **then**
21: Solve VLiM given using $nodesMap[]$
22: **else**
23: Reject Ψ^i
24: **end if**
25: **end procedure**
26: **procedure** MCTS(Tree Node TN, Computational Budget β)
27: **while** $\beta > 0$ **do**
28: $Reward \leftarrow$ SIMULATE (TN)
29: **if** $Reward = \Gamma$ **then**
30: **return** ε
31: **end if**
32: $TN.v \leftarrow TN.v + Reward$
33: $TN.\sigma \leftarrow TN.\sigma + 1$
34: $\beta \leftarrow \beta - 1$
35: **end while**
36: **return** $\underset{i}{\operatorname{argmax}} \left(\frac{TN.child[i].v}{TN.child[i].\sigma}\right)$
37: **end procedure**
38: **procedure** SIMULATE(Tree Node TN)
39: $snI = \underset{i}{\operatorname{argmax}} \left(\frac{TN.child[i].v}{TN.child[i].\sigma} + D\sqrt{\frac{\ln(TN.\sigma)}{TN.child[i].\sigma}}\right)$
40:
41: $(\phi_{next}, Reward) \leftarrow$ SAMPLENEXTSTATE (TN, snI)
42: **if** ϕ_{next} is a terminal state **then**
43: **return** $Reward$
44: **end if**
45: **if** $TN.child[snI].State$ does not exist **then**
46: Create a Tree Node $TN'(v = 0, \sigma = 0, State = \phi_{next})$
47: $TN.child[i] \leftarrow TN'$
48: $Reward \leftarrow$ ROLLOUT $(TN.child[snI])$
49: **else**
50: $Reward \leftarrow$ SIMULATE $(TN.child[snI])$
51: **end if**
52: $TN.child[snI].v \leftarrow TN.child[snI].v + Reward$
53: $TN.child[snI].\sigma \leftarrow TN.child[snI].\sigma + 1$
54: **return** $Reward$
55: **end procedure**
56: **procedure** SAMPLENEXTSTATE(Tree Node TN, snI)
57: $n^{\Psi_i}_{current} \leftarrow TN.State.N^{\Psi_i}.head$
58: **if** $n^s_{snI} \in N^s(n^{\Psi_i}_{current})$ **then**
59: $\phi_{next} \leftarrow TN.State$
60: $\phi_{next}.N^{\Psi_i} \setminus \{n^{\Psi_i}_{current}\}$
61: $\phi_{next}.N^s \setminus \{n^s_{snI}\}$
62: **if** ϕ_{next} is a terminal state **then**
63: Find the current action policy π by traversing the tree from the root to TN
64: Add the action $(n^{\Psi_i}_{current}, n^s_{snI})$ to π
65: Solve VLiM (SP or MCF) using the node mapping π
66: Calculate $Reward$ based on π and the solution of VLiM
67: **return** $(\phi_{next}, Reward)$
68: **else**
69: **return** $(\phi_{next}, 0)$
70: **end if**
71: **else**
72: **return** (ρ, Γ)
73: **end if**
74: **end procedure**
75: **procedure** ROLLOUT(Tree Node TN)
76: $\phi_{current} \leftarrow TN.State$
77: Find the current action policy π by traversing the tree from the root to TN
78: **while** $\phi_{current}$ is not terminal **do**
79: $n^{\Psi_i}_{current} \leftarrow \phi_{current}.N^{\Psi_i}.head$
80: Select a random substrate node $n^s_{current} \in N^s(n^{\Psi_i}_{current})$
81: **if** $n^s_{current} = \varepsilon$ **then**
82: **return** Γ
83: **end if**
84: Add the action $(n^{\Psi_i}_{current}, n^s_{current})$ to π
85: $\phi_{current}.N^{\Psi^i} \setminus \{n^{\Psi^i}_{current}\}$
86: $\phi_{next}.N^s \setminus \{n^s_{current}\}$
87: **end while**
88: Solve VLiM (SP or MCF) using the node mapping π
89: Calculate $Reward$ based on π and the solution of VLiM
90: **return** $Reward$
91: **end procedure**

The GNU Scientific Library random number generator [39] is used to generate random numbers and the necessary probability distributions. We use the MT19937 *Mersenne Twister* [40] random number generator with the default seed 0. The system time is used as the seed for the Standard C Library function `rand()` to generate five additional random seeds in the range of 0 to 1,000: 406, 249, 707, 596, and 778. The GNU Linear Programming Kit (GLPK) [41] is used for solving the MCF problem.

In simulations, we use topology, substrate resources, and VNR requirements that have been adopted in the literature [7], [12], [14]. The Boston University Representative Topology Generator (BRITE) [42] is used to generate the substrate and VNR network graphs. The substrate graph consists of 50 nodes that are randomly placed on a 25×25 Cartesian plane [7]. Connections between the nodes are generated based on the Waxman algorithm [43] with the parameter $\alpha = 0.5$ and the exponential parameter $\beta = 0.2$ [44]. Each substrate node is connected to a maximum of 5 nodes. The generated substrate network graph has 221 edges. The VNR graphs are generated using the same process. The number of nodes in VNR graphs is uniformly distributed between 3 and 10 [7]. Each virtual node is connected to a maximum of 3 virtual nodes.

The CPU capacity of substrate nodes and the bandwidth of substrate links are uniformly distributed between 50 and 100 units. The CPU requirements of virtual nodes are uniformly

Algorithm 2 Pseudocode of parallelized MaVEn algorithm.

procedure PARALLEL_MAVEN($\Psi_i, G^s(N^s, E^s)$, β, p)
  if $my\_rank \in \{0, \ldots, p\}$ **then**
    $\phi \leftarrow (N^{\Psi^i}, N^s)$
    Create $Root$ $(v = 0, \sigma = 0, State = \phi)$
    $nodesMap[] \leftarrow \varnothing$
    $vnI \leftarrow 1$
    do
      $snI \leftarrow$ MCTS $(Root, \beta)$
      if $snI \neq \varepsilon$ **then**
        $nodesMap.Add\ (n^{\Psi_i}_{vnI}, n^s_{snI})$
        if $Root.child[snI].State$ is terminal **then**
          $terminate \leftarrow true$
        else
          $vnI \leftarrow vnI + 1$
          $Root \leftarrow Root.child[snI]$
        end if
      else
        $terminate \leftarrow$ true
      end if
    while $terminate \neq$ true
    if $nodesMap.Size = |N^{\Psi^i}|$ **then**
      $linksMap[] \leftarrow$ VLiM $(G^s, \Psi_i, nodesMap[])$
      Calculate $\mathcal{F}(G^{\Psi_i})$ using $linksMap[]$ and $nodesMap[]$)
      if $my\_rank \neq 0$ **then**
        Send my_rank and $\mathcal{F}(G^{\Psi_i})$ to the master processor $(rank = 0)$
      else
        Receive $\mathcal{F}_m(G^{\Psi_i})$ from processors $rank = m\ \ \forall m \in \{1, \ldots, p\}$
        Identify the $rank_{max}$ of the processor that has the highest $\mathcal{F}(G^{\Psi_i})$
        Receive $linksMap_{max}[]$ and $nodesMap_{max}[]$ from the processor $rank_{max}$
        Send the $linksMap_{max}[]$ and $nodesMap_{max}[]$ to all other processors
      end if
    else
      Reject Ψ^i
    end if
  end if
end procedure

distributed between 2 and 20 units while the bandwidth requirements of virtual links are uniformly distributed between 0 and 50 units [7]. The maximum allowable distance δ for embedding VNR nodes is uniformly distributed between 15 and 25 distance units.

We assume that the VNRs arrivals are a Poisson process with a mean arrival rate of λ requests per unit time. Their life-times are exponentially distributed with a mean $\frac{1}{\mu}$ yielding to a VNR traffic of $\lambda \times \frac{1}{\mu}$ Erlangs. In simulation scenarios, we assume $\frac{1}{\mu} = 1,000$. Duration of each scenario is 50,000 time units [7], [12], [14]. While the simulation time is an abstract clock that ticks 50,000 times, the execution time of the algorithms is based on the clock of the operating system used for execution of the simulation scenarios. Therefore, in order to present consistent and reproducible results that do not depend on the simulation platform, we do not consider the effect of the algorithms' execution time in the simulation scenarios [7].

B. MaVEn Computational Budget Simulation Scenarios

In these simulation scenarios, we assume the constant VNR arrival rate of 2 requests per 100 time units yielding to traffic load of 20 Erlangs. We vary the computational budget β between 5 and 250 samples per virtual node embedding.

Simulation results are shown in Fig. 3. MaVEn algorithms are capable of finding VNE embeddings that are more profitable than those identified by the existing algorithms. The proposed MaVEn-M algorithm achieves the highest acceptance ratio even with a small allocated computational budget β. It further improves the revenue to cost ratio as β increases, which results in higher profitability. With the smallest computational budget $\beta = 5$, the VNE embeddings generated by MaVEn-M are 45% and 65% more profitable than those identified by ViNE and GRC algorithms, respectively. Even though the MaVEn-S algorithm employs a shortest-path based approach for link embeddings, which is stricter than the MCF algorithm employed by the ViNE algorithms, the performance of MaVEn-S algorithm in terms of acceptance ratio is comparable to R-ViNE and D-ViNE algorithms. Furthermore, MaVEn-S finds embeddings with lower revenue to cost ratios, which results in higher profitability. The average processing time of MaVEn-M is considerably higher than the other algorithms. The GRC algorithm achieves the best performance.

C. Variable VNR Arrival Rate Simulation Scenarios

In these simulation scenarios, we assume that the computational budget β is 40 samples per node embedding while the VNR arrival rate is varied between 1 and 8 requests per 100 time units yielding to traffic loads of 10, 20, 30, 40, 50, 60, 70, and 80 Erlangs [14].

Simulation results are shown in Fig. 4. The proposed algorithms achieve better performance compared to the existing algorithms. MaVEn-M has the best acceptance ratio in all scenarios while its revenue to cost ratio drops quickly below the result generated by MaVEn-S at 60 Erlangs. This leads to profitability comparable to MaVEn-S. Because the substrate network resources become scarce at higher VNR traffic loads, there is a higher likelihood of embedding virtual nodes onto substrate nodes that are further apart, which results in more costly embeddings [14]. Although at higher traffic loads the profitabilities of MaVEn-M and MaVEn-S are comparable, MaVEn-M is preferred because it has higher acceptance ratio that may result in higher customer satisfaction.

D. Parallel MaVEn Simulation Scenarios

In these simulation scenarios, we employ root parallelization and execute the MaVEn algorithms in parallel on 2, 4, 6, and 8 processors. We assume that the seed number assigned to each processor is equal to its *rank*. We employ the MPICH2 library [45] that implements the Message Passing Interface (MPI) standard [46]. All simulations were executed on a single Intel Core i7 2600 Quad-Core CPU that employs the Hyper-Threading technology enabling it to execute up to 8 parallel threads. The computational budget and VNR arrival rates are similar to the *Variable VNR Arrival Rate Simulation Scenarios* described in Subsection IV-C.

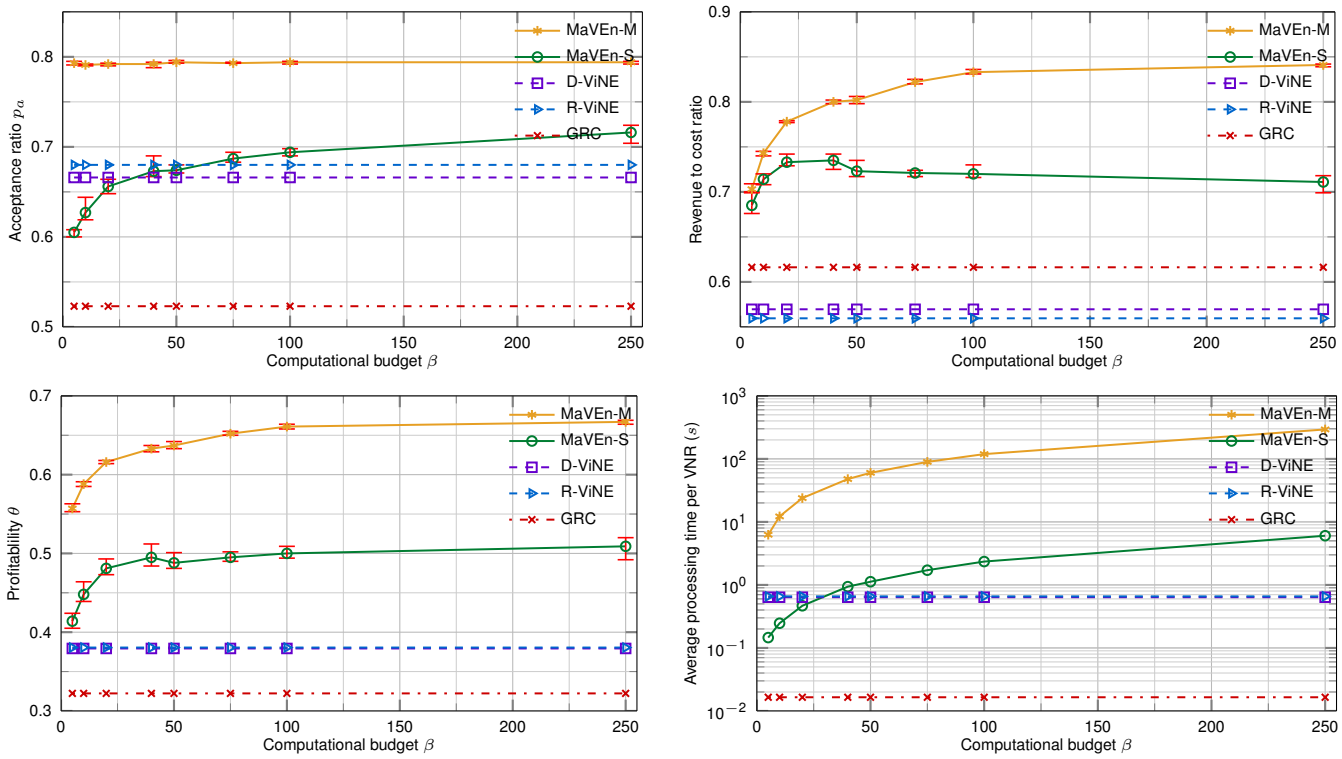

Fig. 3. Comparison of the algorithms with a VNR traffic load of 20 Erlangs. Shown are various performance metrics as functions of computational budget β defined as the number of evaluated action samples per virtual node embedding. The results shown for MaVEn-M and MaVEn-S scenarios are averaged over six executions with randomly generated seeds.

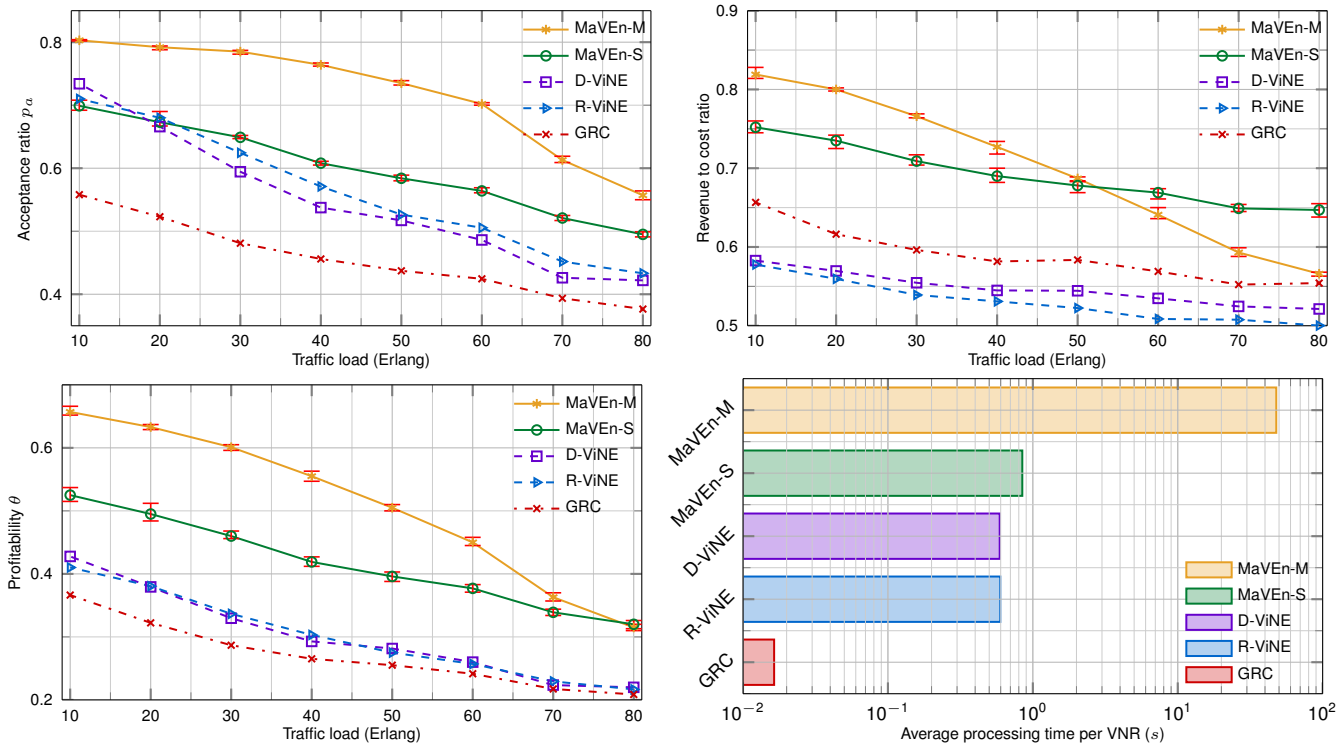

Fig. 4. Performance of the algorithms with various VNR traffic loads. The MaVEn computation budget is $\beta = 40$ samples per virtual node embedding. The results shown for MaVEn-M and MaVEn-S scenarios are averaged over six executions with randomly generated seeds.

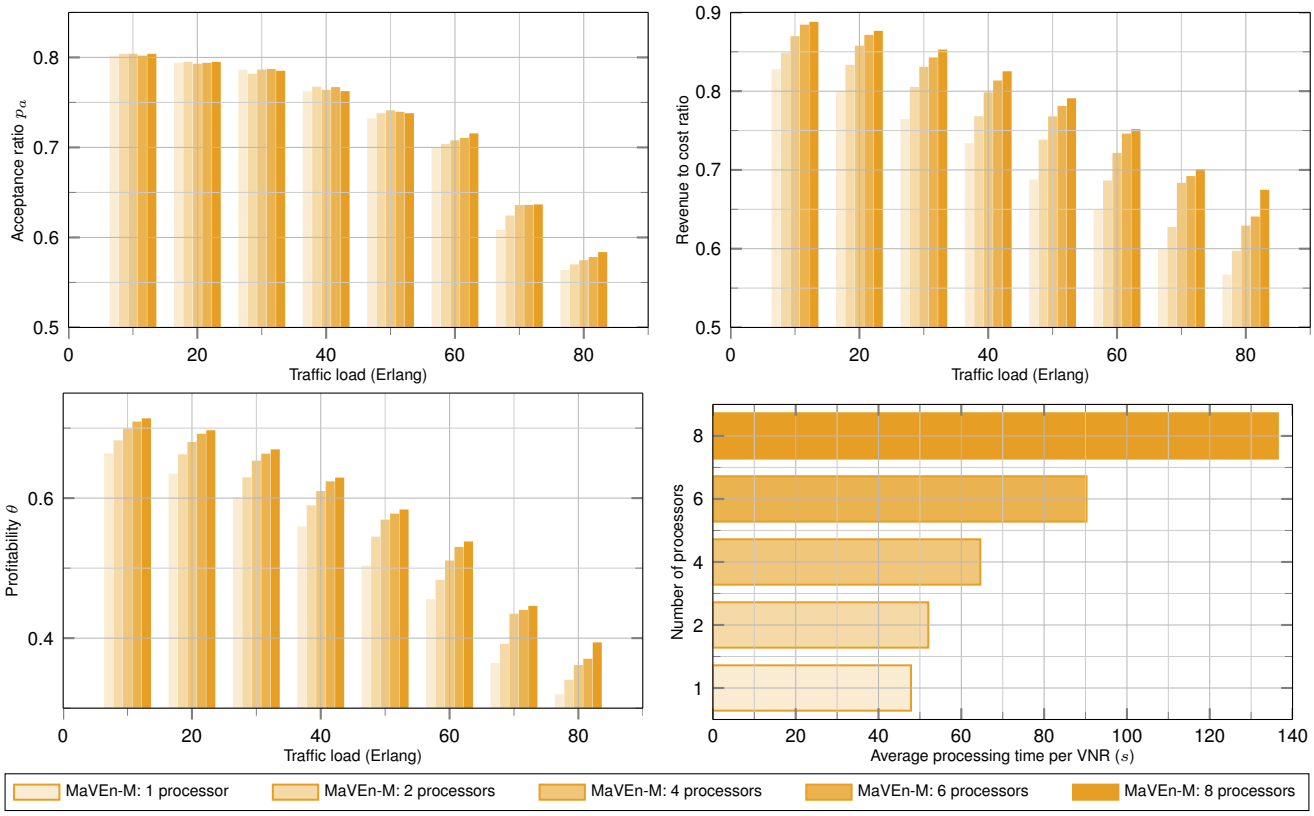

Fig. 5. Performance of the parallel MaVEn-M algorithm with various VNR traffic loads using 1, 2, 4, 6, and 8 processors. The computation budget is $\beta = 40$ samples per virtual node embedding. Shown are various performance metrics as functions of VNR traffic load.

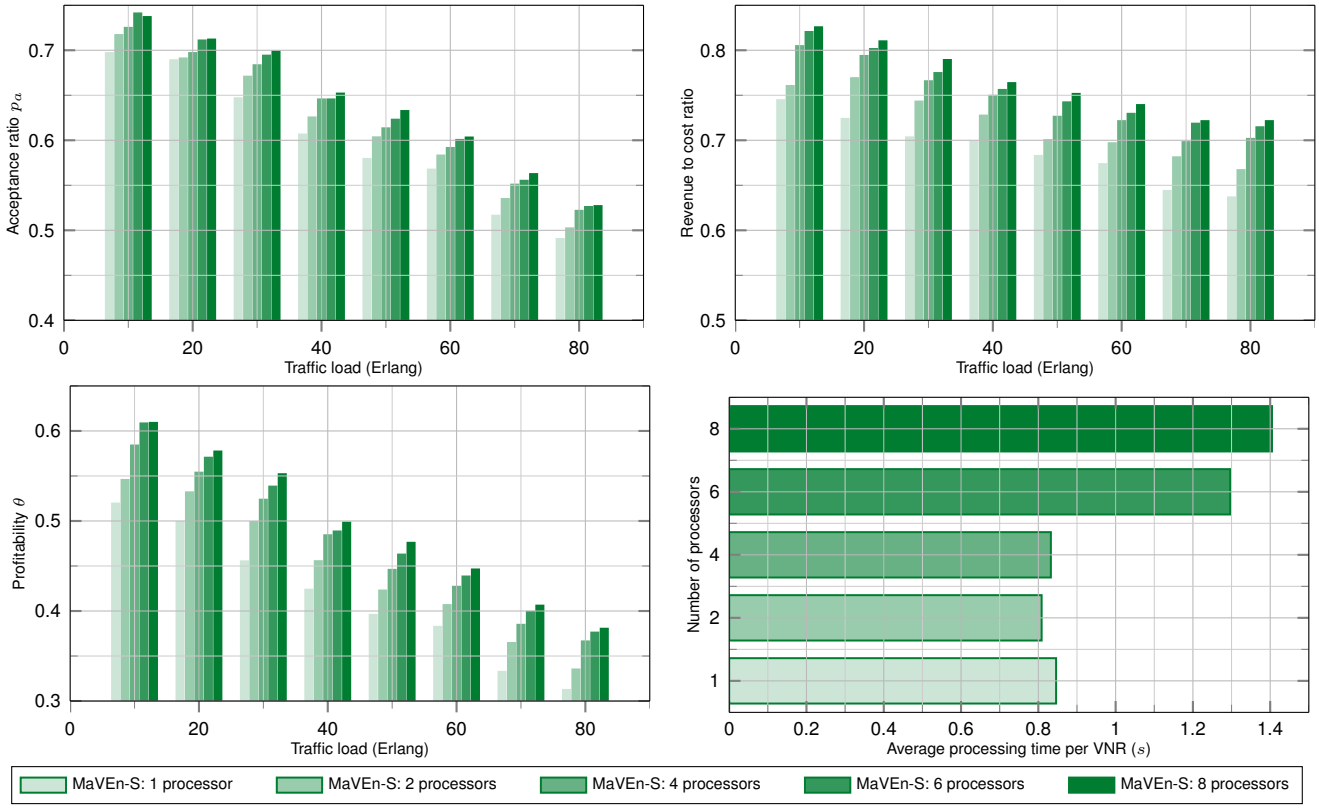

Fig. 6. Performance of the parallel MaVEn-S algorithm with various VNR traffic loads using 1, 2, 4, 6, and 8 processors. The computation budget is $\beta = 40$ samples per virtual node embedding. Shown are various performance metrics as functions of VNR traffic load.

Performance of the parallel MaVEn-M algorithm is shown in Fig. 5. While the parallelization does not have a large impact on the acceptance ratio, it improves the revenue to cost ratio up to 10%. Hence, it results in identifying more profitable virtual network embeddings. The processing time of the algorithms is proportional to the number of used processors. Two factors contribute to the increase in the MaVEn-M processing time: disk I/O operations required by the MCF solver and the communication between processors. The I/O operations have the prevailing effect on the increase in processing time. Since the simulations were executed on a single machine, all processors share a common disk. Therefore, it is impossible to perform the disk I/O operations in parallel. As the number of processors increases, the likelihood of multiple processors simultaneously accessing the disk increases, resulting in higher execution time.

Performance of the parallel MaVEn-S algorithm is shown in Fig. 6. Parallelization improves its acceptance and revenue to cost ratios resulting in increased profitability by up to 10%. The Intel Core i7 2600 Quad-Core CPU used for simulations employs hyper-threading where the operating system views each physical processor (core) as two logical processors. In a hyper-threaded architecture, physical execution resources are shared and the architecture state is duplicated [47]. Parallelization with MPI on a hyper-threaded architecture increases competition for accessing network and for the memory hierarchy resources [48]. Therefore, using up to 4 processors has no impact on the processing time of MaVEn-S because all threads are executed on physical processors. However, processing time of the algorithm increases when utilizing 6 or 8 processors because in these cases hyper-threading is employed.

E. Discussion

The MaVEn-S and GRC algorithms use a shortest-path-based algorithm without path splitting to solve VLiM, which is stricter than the MCF algorithm that enables path splitting utilized by MaVEn-M and ViNE algorithms. For example, consider a virtual node $n_x^{\Psi_i}$ attached to a virtual link that requires 5 units of bandwidth and a substrate node n_y^s attached to two substrate links with available bandwidths of 4 and 1. While utilizing MCF permits embedding $n_x^{\Psi_i}$ onto n_y^s, such embedding is infeasible without path splitting.

Superior performance of the proposed algorithms comes at the cost of higher execution time. The GRC algorithm has the lowest execution time. However, the solutions found are less profitable compared to other algorithms. The stochastic processes governing the VNRs arrival and life-time distributions have not been well investigated [7]. Hence, it is difficult to estimate a reasonable trade-off between execution time and profitability. A five-year traffic analysis (2002 to 2007) of the research-based ProtoGENI project [49] identified an average of 4 VNR arrivals per hour [50]. Based on this low arrival rate, we may assume that allocating a few minutes of processing time is feasible. Even though we have not considered varying the computational budget during simulations, InP operators may adjust the execution time of the MaVEn algorithms according to VNR arrival rates and the size of VNR graphs to avoid long queuing delays.

We have used the GLPK to solve the MCF problem, which currently does not support distributed computing. Furthermore, our implementation of the shortest-path algorithm relies only on the random access memory while the MCF implementation relies on slow disk I/O operations. Parallelizing the MCF and eliminating the disk I/O improves the performance of MaVEn-M and ViNE algorithms.

MCTS parallelization improves performance of the MaVEn algorithms. While we simulated the parallel MaVEn algorithms using multiple threads of a single CPU, they may be executed on clusters that are highly optimized for parallel computing and comprise large number of processors.

V. Conclusion

In this paper, we modeled the Virtual Network Embedding (VNE) problem as a Markov Decision Process (MDP) and proposed two new VNE algorithms (MaVEn-M and MaVEn-S) that solve the proposed MDP by utilizing the Monte Carlo Tree Search. We developed a discrete event simulator VNE-Sim for evaluating VNE algorithms and implemented MaVEn-M, MaVEn-S, R-ViNE, D-ViNE, and GRC algorithms for comparisons. The simulation results show that the proposed algorithms exhibit promising performance. Their advantage is that, time permitting, they search for more profitable embeddings compared to the available algorithms.

References

[1] J. S. Turner and D. E. Taylor, "Diversifying the Internet," in *Proc. IEEE GLOBECOM 2005*, vol. 2, St. Louis, MO, USA, Dec. 2005, pp. 755–760.

[2] T. Anderson, L. Peterson, S. Shenker, and J. S. Turner, "Overcoming the Internet impasse through virtualization," *Computer*, vol. 38, no. 4, pp. 34–41, Apr. 2005.

[3] N. Feamster, L. Gao, and J. Rexford, "How to lease the Internet in your spare time," *Comput. Commun. Rev.*, vol. 37, no. 1, pp. 61–64, Jan. 2007.

[4] N. M. M. K. Chowdhury and R. Boutaba, "Network virtualization: state of the art and research challenges," *IEEE Commun. Mag.*, vol. 47, no. 7, pp. 20–26, July 2009.

[5] A. Belbekkouche, M. M. Hasan, and A. Karmouch, "Resource discovery and allocation in network virtualization," *IEEE Communications Surveys & Tutorials*, vol. 14, no. 4, pp. 1114–1128, 2012.

[6] Y. Zhu and M. Ammar, "Algorithms for assigning substrate network resources to virtual network components," in *Proc. IEEE INFOCOM*, Barcelona, Spain, Apr. 2006, pp. 1–12.

[7] M. Chowdhury, M. R. Rahman, and R. Boutaba, "ViNEYard: Virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM Trans. Netw.*, vol. 20, no. 1, pp. 206–219, Feb. 2012.

[8] D. G. Andersen, "Theoretical approaches to node assignment," Dec. 2002, Unpublished Manuscript. [Online]. Available: http://repository.cmu.edu/compsci/86/.

[9] M. You, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *Comput. Commun. Rev.*, vol. 38, no. 2, pp. 19–29, Mar. 2008.

[10] I. Houidi, W. Louati, W. Ben Ameur, and D. Zeghlache, "Virtual network provisioning across multiple substrate networks," *Computer Networks*, vol. 55, no. 4, pp. 1011–1023, Mar. 2011.

[11] J. F. Botero, X. Hesselbach, M. Duelli, D. Schlosser, A. Fischer, and H. de Meer, "Energy efficient virtual network embedding," *IEEE Commun. Lett.*, vol. 16, no. 5, pp. 756–759, May 2012.

[12] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, and J. Wang, "Virtual network embedding through topology-aware node ranking," *Comput. Commun. Rev.*, vol. 41, pp. 38–47, Apr. 2011.

[13] S. Zhang, Y. Qian, J. Wu, and S. Lu, "An opportunistic resource sharing and topology-aware mapping framework for virtual networks," in *Proc. IEEE INFOCOM*, Orlando, FL, USA, Mar. 2012, pp. 2408–2416.

[14] L. Gong, Y. Wen, Z. Zhu, and T. Lee, "Toward profit-seeking virtual network embedding algorithm via global resource capacity," in *Proc. IEEE INFOCOM*, Toronto, ON, Canada, Apr. 2014, pp. 1–9.

[15] J. Lischka and H. Karl, "A virtual network mapping algorithm based on subgraph isomorphism detection," in *Proc. ACM VISA*, Barcelona, Spain, Aug. 2009, pp. 81–88.
[16] Z. Zhang, X. Cheng, S. Su, Y. Wang, K. Shuang, and Y. Luo, "A unified enhanced particle swarm optimization-based virtual network embedding algorithm," *Int. J. Commun. Syst.*, vol. 26, no. 8, pp. 1054–1073, Aug. 2012.
[17] I. Fajjari, N. Aitsaadi, G. Pujolle, and H. Zimmermann, "VNE-AC: Virtual network embedding algorithm based on ant colony metaheuristic," in *Proc. IEEE ICC 2011*, Kyoto, Japan, June 2011, pp. 1–6.
[18] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: a survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.
[19] A. Razzaq and M. Rathore, "An approach towards resource efficient virtual network embedding," in *Proc. INTERNET 2010*, Valencia, Spain, Sept. 2010, pp. 68–73.
[20] H. Yu, V. Anand, C. Qiao, H. Di, and X. Wei, "A cost efficient design of virtual infrastructures with joint node and link mapping," *J. Netw. and Syst. Management*, vol. 20, no. 1, pp. 97–115, Sept. 2012.
[21] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *Lecture Notes in Computer Science: Proc. 17th European Conference on Machine Learning (ECML)*, J. Furnkranz, T. Scheffer, and M. Spiliopoulou, Eds. Springer, 2006, vol. 4212, pp. 282–293.
[22] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo Tree Search," in *Proc. 5th Int. Conf. Comput. and Games (CG'06)*, Turin, Italy, May 2006, pp. 72–83.
[23] A. M. Uhrmacher, "Dynamic structures in modeling and simulation: a reflective approach," *ACM Trans. Modeling and Computer Simulation*, vol. 11, no. 2, pp. 206–232, Apr. 2001.
[24] J. J. Nutaro, *Building Software for Simulation: Theory and Algorithms, with Applications in C++*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2010.
[25] C. Szepesvári, "Reinforcement learning algorithms for MDPs—a survey," Department of Computing Science, University of Alberta, Tech. Rep. TR09-13, 2009.
[26] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, Wiley Series in Probability and Statistics. Hoboken, NJ, USA: John Wiley & Sons, Inc., 1994.
[27] M. L. Littman, T. L. Dean, and L. P. Kaelbling, "On the complexity of solving Markov decision problems," in *Proc. Eleventh Conf. Uncertainty in Artificial Intell.*, Montreal, QU, Canada, Aug. 1995, pp. 394–402.
[28] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1998.
[29] M. Kearns, Y. Mansour, and A. Y. Ng, "A sparse sampling algorithm for near-optimal planning in large Markov decision processes," *Machine Learning*, vol. 49, no. 2, pp. 193–208, 2002.
[30] H. Baier and M. H. M. Winands, "Nested Monte-Carlo Tree Search for online planning in large MDPs," in *Proc. 20th European Conf. Artificial Intelligence*, Montpellier, France, Aug. 2012, pp. 109–114.
[31] D. Silver and J. Veness, "Monte-Carlo planning in large POMDPs," in *Advances in Neural Inform. Process. Syst. 23: 24th Annual Conference on Neural Information Processing Systems*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, Eds. Curran Associates, Inc., 2010, vol. 3, pp. 2164–2172.
[32] P. P. Shenoy, "Game trees for decision analysis," *Theory and Decision*, vol. 44, no. 2, pp. 149–171, Apr. 1998.
[33] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: a survey," *J. of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.
[34] S. Gelly and D. Silver, "Monte-Carlo tree search and rapid action value estimation in computer Go," *Artificial Intelligence*, vol. 175, no. 11, pp. 1856–1875, July 2011.
[35] M. P. D. Schadd, M. H. M. Winands, M. J. W. Tak, and J. W. H. M. Uiterwijk, "Single-player Monte-Carlo tree search for SameGame," *Knowledge-Based Systems*, vol. 34, pp. 3–11, Oct. 2012.
[36] G. Chaslot, M. Winands, and H. J. Herik, "Parallel Monte-Carlo tree search," in *Lecture Notes in Computer Science: Computers and Games*, H. J. Herik, X. Xu, Z. Ma, and M. Winands, Eds. Springer, 2008, vol. 5131, pp. 60–71.
[37] T. Cazenave and N. Jouandeau, "On the parallelization of UCT," in *Proc. Computer Games Workshop 2007 (CGW 2007)*, H. J. van den Herik, J. W. Uiterwijk, and M. H. Winands, Eds. Universiteit Maastricht, 2007, vol. 5131, pp. 93–101.
[38] (2016, Jan.) Vne-sim repository. [Online]. Available: https://bitbucket.org/shaeri/vne-sim/.
[39] (2016, Jan.) GSL–GNU Scientific Library. [Online]. Available: https://www.gnu.org/software/gsl/.
[40] M. Matsumoto and T. Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudorandom number generator," *ACM Trans. Modeling and Computer Simulation*, vol. 8, no. 1, pp. 3–30, Jan. 1998.
[41] (2016, Jan.) GLPK–GNU Linear Programming Kit. [Online]. Available: http://www.gnu.org/software/glpk/.
[42] (2016, Jan.) Boston university representative internet topology generator. [Online]. Available: http://www.cs.bu.edu/brite/.
[43] B. M. Waxman, "Routing of multipoint connections," *IEEE J. Sel. Areas Commun.*, vol. 6, no. 9, pp. 1617–1622, Dec. 1988.
[44] E. W. Zegura, K. L. Calvert, and M. J. Donahoo, "A quantitative comparison of graph-based models for Internet topology," *IIEEE/ACM Trans. Netw.*, vol. 5, no. 6, pp. 770–783, Dec. 1997.
[45] (2015, Dec.) MPICH: High-Performance Portable MPI. [Online]. Available: https://www.mpich.org/.
[46] (2015, Dec.) Message Passing Interface Forum. [Online]. Available: http://www.mpi-forum.org/.
[47] D. T. Marr, F. Binns, D. L. Hill, G. Hinton, D. A. Koufaty, J. A. Miller, and M. Upton, "Hyper-Threading technology architecture and microarchitecture," *Intel Technol. J.*, vol. 6, no. 1, pp. 4–15, Feb. 2002.
[48] S. Saini, H. Jin, R. Hood, D. Barker, P. Mehrotra, and R. Biswas, "The impact of hyper-threading on processor resource utilization in production applications," in *Proc. 18th Int. Conf. High Performance Computing (HiPC 2011)*, Bengaluru, India, Dec. 2011, pp. 18–21.
[49] (2015, July) ProtoGENI. [Online]. Available: http://www.protogeni.net/.
[50] Q. Yin and T. Roscoe, "VF2x: Fast, efficient virtual network mapping for real testbed workloads," in *Proc. 8th Int. ICST TridentCom 2012*, Thessaloniki, Greece, June 2012, pp. 271–286.

Soroush Haeri (S'11) received the undergraduate degree from Multimedia University, Cyberjaya, Malaysia. He is currently pursuing the Ph.D. degree in the School of Engineering Science, Simon Fraser University, Burnaby, BC, Canada.

He is a member of the Communication Networks Laboratory at Simon Fraser University. He was a Software Engineer with Wavelet Solutions, Subang Jaya, Malaysia, from 2010 to 2011. In 2013, he worked as a developer of Android applications at Simon Fraser University Radio Station CJSF, Burnaby, BC, Canada. His current research interests include communication networks, applications of machine-learning algorithms to routing in computer networks, and scalable routing architectures.

Ljiljana Trajković received the Dipl. Ing. degree from University of Pristina, Yugoslavia, in 1974, the M.Sc. degrees in electrical engineering and computer engineering from Syracuse University, Syracuse, NY, in 1979 and 1981, respectively, and the Ph.D. degree in electrical engineering from University of California at Los Angeles, in 1986.

She is currently a Professor in the School of Engineering Science at Simon Fraser University, Burnaby, British Columbia, Canada. From 1995 to 1997, she was a National Science Foundation (NSF) Visiting Professor in the Electrical Engineering and Computer Sciences Department, University of California, Berkeley. She was a Research Scientist at Bell Communications Research, Morristown, NJ, from 1990 to 1997, and a Member of the Technical Staff at AT&T Bell Laboratories, Murray Hill, NJ, from 1988 to 1990. Her research interests include high-performance communication networks, control of communication systems, computer-aided circuit analysis and design, and theory of nonlinear circuits and dynamical systems.

Dr. Trajković serves as Junior Past President (2016 – 2017) of the IEEE Systems, Man, and Cybernetics Society and served as President (2014 – 2015), President-Elect (2013), Vice President Publications (2012 – 2013 and 2010 – 2011), Vice President Long-Range Planning and Finance (2008 – 2009), and a Member at Large of its Board of Governors (2004 – 2006). She served as 2007 President of the IEEE Circuits and Systems Society. She was a member of the Board of Governors of the IEEE Circuits and Systems Society (2001 – 2003 and 2004 – 2005). She is Chair of the IEEE Circuits and Systems Society joint Chapter of the Vancouver/Victoria Sections. She was Chair of the IEEE Technical Committee on Nonlinear Circuits and Systems (1998). She is General Co-Chair of SMC 2016 and served as General Co-Chair of HPSR 2014, Technical Program Co-Chair of ISCAS 2005, and Technical Program Chair and Vice General Co-Chair of ISCAS 2004. She served as an Associate Editor of the IEEE Transactions on Circuits and Systems (Part I) (2004 – 2005 and 1993 – 1995), the IEEE Transactions on Circuits and Systems (Part II) (1999 – 2001 and 2002 – 2003), and the IEEE Circuits and Systems Magazine (2001 – 2003). She was a Distinguished Lecturer of the IEEE Circuits and Systems Society (2010 – 2011 and 2002 – 2003). She is a Professional Member of IEEE-HKN and a Fellow of the IEEE.