

NOTICE

This work has been submitted to the IEEE for possible publication. Copyright may be transferred without notice, after which this version may no longer be accessible.

Isolation Scheme for Virtual Network Embedding Based on Reinforcement Learning for Smart City Vertical Industries

Ali Gohar

Department of Electrical Engineering and Computer Science, University of Stavanger, Stavanger, Norway
ali.gohar@uis.no

Abstract—Modern ICT infrastructure is built on virtualization technologies, which connect a diverse set of dedicated networks to support a variety of smart city vertical industries (SCVI), such as energy, healthcare, manufacturing, entertainment, and intelligent transportation. The wide range of SCVI use cases require services to operate continuously and reliably. The violation of isolation by a specific SCVI, that is, a SCVI network must operate independently of other SCVI networks, complicates service assurance for infrastructure providers (InPs) significantly. As a result, a solution must be considered from the standpoint of isolation, which raises two issues: first, these SCVI networks have diverse resource requirements, and second, they necessitate additional functionality requirements such as isolation. Based on the above two problems faced by SCVI use cases, we propose a virtual network embedding (VNE) algorithm with resource and isolation constraints based on deep reinforcement learning (DRL). The proposed DRL_VNE algorithm can automatically adapt to changing dynamics and outperforms existing three state-of-the-art solutions by 12.9%, 19.0% and 4% in terms of the acceptance rate, the long-term average revenue, and long-term average revenue to cost ratio.

Index Terms—Internet of Things, Isolation, Reinforcement Learning, Resource Allocation, Smart City, Virtual Network Embedding, Vertical Industries

I. INTRODUCTION

The fifth-generation (5G) mobile network provides differentiated services over a shared networking infrastructure and plays a key role to support services required by various smart city vertical industries (SCVIs), including energy, health care, manufacturing, media and entertainment, automotive, and intelligent transportation [1]. Enabled by paradigms, such as Network Function Virtualization (NFV), and network slicing allows Infrastructure Providers (InPs) to create an isolated, end-to-end logical network slices or Virtual Network (VN) which are composed of a set of interconnected VNFs.

In order to provide services to their users, the Service Providers (SPs) request that InP create customized VNs for them. InP deploys the necessary VNFs to create a VN for SP on the infrastructure, which consists of heterogeneous resources such as computing, storage, and a network, also known as a Substrate Network (SN).

Efficient mapping the network slice towards the InP or VNs to the SN primarily deals with the VNF placement or is a typical Virtual Network Embedding (VNE) problem which

has been intensively studied in the literature with various perspectives such as resource allocation, privacy and security [2]. Moreover, various algorithms have been proposed to solve the VNE problem and are broadly categorized in to three categories heuristic, exact and DRL [2] [3].

However, less attention has been paid by the researchers on the challenges that SCVIs face concerning isolation which is the property that one VN operates without any influence of other VN utilizing the same infrastructure [4]. SCVIs have a wide range of critical use cases, so they must operate continuously and reliably. The violation of isolation by a specific SCVI VN significantly complicates service assurance for the InPs because it is difficult to identify the root causes of performance degradation due to failures or security breaches that may propagate from the original VN to other VNs. In order to keep pace with the rapid advancement of new technologies, InPs must take isolation into account from the perspective of the SN architecture.

In this paper, we propose a VNE algorithm based on deep reinforcement learning (DRL) which has performed well on a set of complicated sequential decision-making problems, such as the game of Dota 2 [5], Hide & Seek [6]. Our proposed a VNE algorithm based on DRL considers computing, storage and isolation of three dimensional resource constraints. The paper has the following main contributions:

- We describe the VNE allocation problem constrained by computation, storage and isolation attributes;
- Our proposed solution is based on DRL which is adaptive, online, and can deploy VNs with varying requirements.
- We compare the performance of the proposed solution with three existing works [7] [8] [9].

The rest of this paper is organized as follows: Section II describes the problem model. Section III describes the solution algorithm. Section IV compares the performance of the proposed algorithm with the state-of-the-art ones, and finally, Section V concludes this study.

II. PROBLEM MODEL AND DESCRIPTION

In this section, we describe the actors involved; the model of SN and VN; the formal definition of VNE problem and finally, the objective of our problem.

In this problem, two actors are considered, InPs and SP. The InPs own resources and create VNs for the SPs after receiving a Virtual Network Request (VNR). In the next two subsections we present the type of resources provided by InPs and then the requirements for a VNRs.

A. SN Model

We consider the InP SN consisting of multidimensional resources such as computing, storage, and networking that are owned or managed by various InPs. The InP SN is composed of two elements as detailed below:

- 1) *Computational platforms*: These are the virtual resources in Computing Platforms (CPs). We indicate N^S as the set of CPs where A_N^S represents a configuration for each CP. For each CP configuration, we consider the following resources:

- the amount of processing $CPU(n^s)$ in [vCPU];
- the amount of storage $STO(n^s)$ in [GiB];
- isolation level $ISL(n^s)$ in [Units]; $\forall n^s \in N^S$.

- 2) *Logical connections*: These are network resources and are provided by network operators. The Logical Connections (LCs) connect CPs. We indicate L^S as the set of LCs where A_L^S represents a configuration for each LC. We consider the following resources for each LC configuration:

- the amount of data rate $BW(l^s)$ in [Mb/s]; $\forall l^s \in L^S$

The SN model can now be formalized using graph theory and can be represented by $G^S = \{N^S, L^S, A_N^S, A_L^S\}$.

B. VNR Model

A VNR in our system is defined network services that supports various SVCI use cases which are characterized by a predominant requirements of bandwidth, latency or massive number of connections. We focus on allocating resources in the edge and core networks and not in the access network (i.e. radio resources).

- 1) *Computing requirements*: We indicate N^V as the set of VNFs where A_V^S represents a configuration for each VNF. For each VNF configuration, we consider the following resources:

- computing resource requirement $CPU(n^v)$, in [vCPU];
- storage resource requirement $STO(n^v)$ in [GiB];
- isolation level requirement $ISL(n^v)$ in [Units]; $\forall n^v \in N^V$.

- 2) *Networking requirement*: Since, VNFs are connected together. We consider virtual links that connect various VNFs together. We indicate L^V as the set of LCs where A_L^V represents a configuration for each LC. We consider the following resources for each LC configuration:

- data rate requirement $BW(l^v)$ in [Mb/s]; $\forall l^v \in L^V$

- 3) *Arrival and departure of VNR*: It is worth noting that when an VNR is allocated, the deletion time of an VNR is uncertain. We use t to denote the arrival/creation time of a VNR, and t_d to denote the departure/deletion time of the VNR.

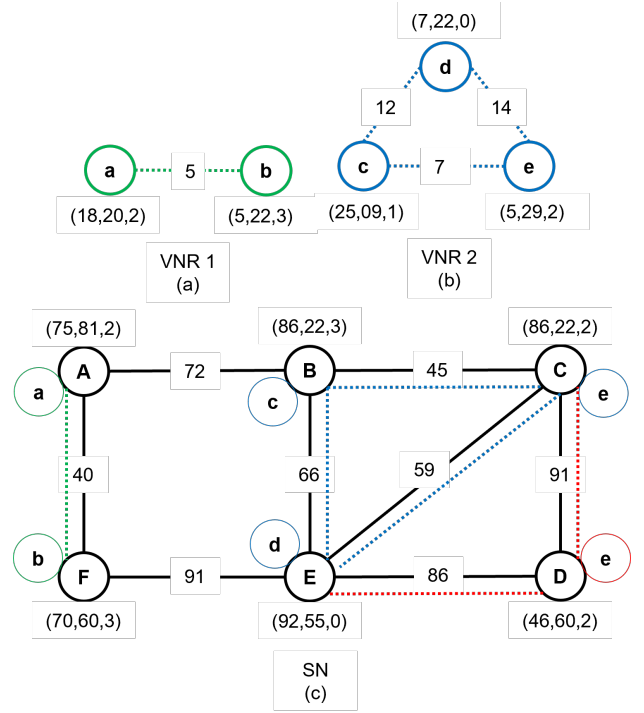


Fig. 1. SN & VN model with possible embedding examples

The VNR model is formalized using graph theory and is represented as $G^V = \{N^V, L^V, A_n^V, A_L^V\}$.

C. VNE Problem Description

The VNE problem can be defined as a mapping M from G^V to G^S : $G^V(N^V, L^V) \rightarrow G^S(N', P')$ where $N' \subset N^S, P' \subset P^S$. The main objective is to find a solution that satisfies VNR requirements while allocating different SN resource types.

Figure 1, (a) and (b) depict two distinct VNRs, (VNR 1 and VNR 2) while (c) depicts an SN. The numbers in brackets next to the virtual nodes indicate the computing, storage, and security level requirements for two VNRs. The bandwidth needs of the virtual links are indicated by the rest numbers on those links. Similarly, the numbers in brackets in SN represent the available computing resources, storage resources, and security levels, respectively. The rest numbers represent the available bandwidth resources.

We highlight three possible scenarios in which VNRs are embedded in the SN. The virtual nodes a and b in VNR 1 are mapped to the CPs A and F. However, the virtual nodes c, d, and e in VNR 2 can be mapped to the CPs B, C, and E or to the CPs C, D, and E, respectively. It should be noted that the virtual node e of VNR 2 can be embedded in either CP C or D; however, embedding virtual node e of VNR 2 in CP D will use consume more link bandwidth resources and is not an optimal allocation.

D. Optimization Objectives

Virtual network requests arrive with unknown & varying time distributions and resource requirements. The main goal

of VNE algorithm is to maximize ISP revenue by making efficient use of substrate network resources. In this paper, we look at two important metrics that have been widely used to assess the effectiveness of a VNE algorithm: long-term average revenue and acceptance ratio.

- 1) *Acceptance Rate*: Each VNR should be explicitly accepted or rejected when it arrives at time t . Virtual nodes and links must be mapped to physical nodes and paths with adequate resource capacities in order for a VNR to be accepted. A VNE algorithm that achieves a high acceptance ratio is desirable.

$$ACR = \lim_{T \rightarrow \infty} \frac{\sum_{t=0}^T \text{accepted}(G^V, t)}{\sum_{t=0}^T \text{arrive}(G^V, t)} \quad (1)$$

Eq. 1 shows the acceptance rate where $\text{accepted}(G^V, t)$ denotes the total number of VNRs that are successfully embedded and $\text{arrive}(G^V, t)$ represents the total number of VNRs that make resource requests.

- 2) *Long-term Revenue to Cost Ratio*: The amount of revenue generated by an InP for accepting a VNR is determined by the amount of resources requested by the VNR and its duration. Allowing SP to use more InP resources to generate more revenue is preferable for InP.

$$REV(G^V) = \sum_{i=1}^{|N^V|} [CPU(n_i^v) + STO(n_i^v)] + \sum_{j=1}^{|L^V|} BW(l_j^v) \quad (2)$$

Eq. 2, shows the revenue of InP, where $CPU(n_i^v) + STO(n_i^v)$ represents the computing and storage resources occupied by the virtual node (n_i^v). $BW(l_j^v)$ represents the bandwidth resource occupied by the virtual link l_j^v .

Eq. 3, shows the cost to InP for providing resource services to the SP and is formulated as follows:

$$CST(G^V) = \sum_{i=1}^{|N^V|} [CPU(n_i^v) + STO(n_i^v)] + \sum_{j=1}^{|L^V|} \sum_{k=1}^{|L^S|} BW(l_{jk}^{vs}) \quad (3)$$

where $CPU(n_i^v) + STO(n_i^v)$ represents the computing and storage resources occupied by the virtual node (n_i^v) after successful embedding. $BW(l_{jk}^{vs})$ represents the bandwidth resource occupied by the virtual link l_{jk}^{vs} .

Eq. 4 defines the long-term revenue to cost ratio given the equations Eq. 2 and 3 as follows:

$$LRC = \lim_{T \rightarrow \infty} \frac{\sum_{t=0}^T REV(G^V, t)}{\sum_{t=0}^T CST(G^V, t)} \quad (4)$$

Eq. 4, show the ratio of long-term average revenue to long-term average cost.

III. ALGORITHM IMPLEMENTATION

Several key elements must be determined before using the RL algorithm. As a result, in this section, we describe SN

node attribute selection, feature matrix and policy network.

A. Attribute of SN

We use neural network to build a policy network as the learning agent with SN node features as input. The node features are normalized and serve as a training environment for the agent. Due to increased computational complexity, extracting all node features would be unrealistic, and insufficient feature extraction would result in an inadequate representation of the environment. We extract the following four features for each SN node:

- $CPU(n^s)$: The CPU capacity of an SN node has a significant impact on its availability.
- $STO(n^s)$: Using storage capacity can improve the efficiency of underlying resources while decreasing bandwidth demand.
- $ISL(n^s)$: Setting an appropriate isolation level will ensure the isolation of resource scheduling for the SVC use cases.
- Average Distance (AVD): By selecting an SN node close to those that have already been mapped, the cost of SN link bandwidth can be reduced. As a result, we define the AVD as the distance between an SN node and the rest of the mapped SN nodes, which is calculated using the Floyd-Warshall [10] algorithm and is given as follows:

$$AVD = \frac{\sum_{i=1, n_i^s \in N^S} \text{dis}(n^s, n_i^s)}{\text{count} + 1} \quad (5)$$

Where $\sum_{i=1, n_i^s \in N^S} \text{dis}(n^s, n_i^s)$ is the distance from the substrate node n^s to the mapped substrate node n_i^s . count is the number of nodes that have been mapped, plus 1 is to prevent the denominator from being 0.

B. Feature Matrix

After the node features are extracted, they need to be normalized to the feature vector. For the k -th bottom node, its feature vector is as follows:

$$V_{n_k^s} = [CPU(n_k^s), STO(n_k^s), ISL(n_k^s), AVD(n_k^s)] \quad (6)$$

Each substrate node has a corresponding feature vector, which is connected in sequence to form a four-dimensional feature matrix:

The purpose of normalization is to accelerate the training process and enable the agent to converge quickly.

$$M_f = [v_{n_1^s}, v_{n_2^s}, \dots, v_{n_{|N^S|}^s}] \quad (7)$$

The policy network receives the feature matrix as an input, and is updated periodically to reflect changes in the substrate network. In our work, we use the feature matrix under the assumption that all VNRs have an invariant distribution. Therefore, if an embedding algorithm performs well on historical VNRs, it is likely to perform similarly on online VNRs.

C. Policy Network

Deep Learning (DL) is a subset of machine learning (ML) that uses an Artificial Neural Network (ANN). In DL, the term “deep” refers to the number of layers in a neural network; it is also known as Deep Neural Network (DNN). Reinforcement Learning (RL) is a machine learning technique that involves learning to make optimal decisions through repeated interactions in an unknown, dynamic environment [11]. As a result, DRL emerged as an ML sub-field combining DL and RL.

In this paper, we use a four-layer ANN to construct a policy network as the learning agent, which takes the feature matrix as input and outputs the probabilities of mapping VNR nodes to SN nodes. The policy network is composed of four layers: the input layer, the convolution layer, the probability layer, and the selection layer.

The input layer receives a feature matrix and delivers it to the convolution layer. The convolution layer is a part of convolution neural network and a class of ANN. It applies a convolution operation to the feature matrix to create a vector that depicts the resources that are available to each node in an SN which is given by Eq. 8.

$$r_i = \omega \cdot v_{n_i} + b \quad (8)$$

r_i is the i th output of the convolution layer, ω is the weight of the convolution kernel vector, and b is the deviation also known as offset or bias. A function of the probability layer is the normalization of the gradient logarithm of the discrete probability distribution of finite terms. It makes use of a normalized exponential function, such as softmax, which is a generalization of logistic regression. Softmax function outputs a probability distribution over all substrate nodes being mapped which is proportional to the probability based on the available resource vector for each substrate node and is given by Eq. 9.

$$P_i = \frac{e^{r_i}}{\sum_k e^{r_n}} \quad (9)$$

where P_i represents the probability that the i th physical node is embedded. n is the number of feature vectors.

Some of the nodes are unable to host the virtual node in question due to a lack of resources and an isolation level. As a result, the selection layer’s function is to filter out the set of candidate substrate nodes with sufficient resource capacities.

D. Training and Testing

To ensure that the learning agent makes the correct decision during the training process, a reward signal must be sent to the policy network. The long-term average revenue-to-cost ratio is used as a reward because it forces the learning agent to make cost-effective decisions by accounting for both the revenue and the cost of the VNE embedding. This reward enables the learning agent to reflect how the underlying network resources are used. Furthermore, a high revenue-to-consumption ratio indicates that the learning agent is effective and should keep making current decisions in order to accumulate rewards.

To learn policy network parameters, we employ the gradient update method. To optimize the parameters of the policy network, the gradient update procedure uses back-propagation to update the loss function in the direction of gradient descent. It simultaneously optimizes the reward mechanism and the policy network, and the update is as follows:

$$\mathcal{J} = \mathcal{J} - \mathcal{J}' \times \alpha \times \text{Reward} \quad (10)$$

α is an important factor that influences the gradient update and training speed. A small value will slow down the training process. A large value will result in bad decisions, and a too large value will result in unstable training. We chose the batch updating approach to ensure learning stability and expedite the process of updating the policy network parameters.

Algorithm 1 shows the training process of the VNE algorithm based on DRL. In the training phase, all parameters of the policy network are randomly initialized in line 1 and is trained for several epochs in line 2. Each VNR is mapped on SN in two stages. In first stage from lines 5–8 VNR nodes are mapped on SN. If all VNR nodes are mapped then in second stage in line 10 the virtual links of VNR nodes are mapped using breadth-first search.

In line 3, the VNR is chosen from the *trainingSet*, and the policy network is given the M_f of SN. In line 7, the policy network creates a list of potential SN nodes and their probabilities that can map to VNR node. If there are no SN nodes or SN links with enough resources, the mapping fails. In line 8, the gradient is calculated. Following the successful mapping of VNR nodes and links the reward is calculated in line 11. During policy network testing, we use greedy strategy to select the SN node with the highest probability as the mapping node for the VNR virtual node. Algorithm 2 shows the testing process of the VNE algorithm based on DRL.

IV. EVALUATION

In this section we first describe the environment simulation parameters for generating SN topology and VNRs based on similar parameters presented in related works [7] [8] [9]. Moreover, the main experiment simulation parameters are also presented in Table. I.

A. Generating SN

Using the GT-ITM tool [12], we create an SN that is roughly the size of a middle-sized ISP and has 110 nodes and about 550 links. The computing capacity of every SN node and the bandwidth of every SN node link is a real number that follows a uniform distribution and is between 50 and 100.

B. Generating VNRs

The VNR arrive with an average of 4 VNRs per 100 time units according to a Poisson distribution, and they stay for the duration with an average of 1000 time units according to an exponential distribution. There are 2–10 virtual nodes in each VNR. Every virtual node has a uniformly distributed computing and bandwidth requirement between 0 and 50 units. We create a timeline with approximately 50,000 time units and

Algorithm 1: Policy Network Training

Input : SN, trainingSet, numEpoch, α , batch_size

```

1 Initialize all policy network parameters
2 while iteration  $\leq$  numEpoch do
3   for VNR  $\in$  trainingSet do
4     count = 0
5     for VNR_node  $\in$  VNR do
6       get Feature Matrix
7       get SN Node Probability Distribution
8       compute Gradient
9     if isMapped ( $\forall$  VNR_nodes  $\in$  VNR)
10      if bfs_linkMapping(VNR)
11        reward = calculateReward(VNR)
12        updateGradient(reward,  $\alpha$ )
13      else
14        clearGradient
15      count++ = 1
16      if count == batch_size
17        count == 0
18  iteration++ = 1

```

Result: return policy network parameters

Algorithm 2: Policy Network Testing

Input : SN, testSet

```

1 Initialize all policy network parameters
2 for VNR  $\in$  testSet do
3   for VNR_node  $\in$  VNR do
4     get Feature Matrix
5     get SN nodeMapping Probability
6   if isMapped ( $\forall$  VNR_nodes  $\in$  VNR )
7     if bfs_linkMapping(VNR)
8       return embeddingSuccess

```

Result: Two evaluation objectives

2000 VNRs, which are equally divided into two sets: a training set and a testing set.

C. DRL-VNE Algorithm

TensorFlow [13] is used to build policy network and Xavier [14] to initialize the parameters of the policy network using normal distribution. We use stochastic gradient descent optimizer to calculate the gradient of each training iteration with a learning rate of 0.005 for 100 epochs. Table. II lists the key parameters of the DRL-based learning agent to reproduce our results.

D. Evaluation Algorithms

We implement our algorithm in Python and use Pycharm as IDE. We compare our algorithm to two node ranking-based heuristic algorithms, including, rw_rank [7], grc_rank [8], and one that uses basic RL mts_vne [9]. Table. III lists the comparison algorithms and their description.

TABLE I
SIMULATION ENVIRONMENT PARAMETERS

Configuration Attribute	Attribute Value
SN nodes	110
Computing resources	U[50,100]
Storage resources	U[50,100]
Isolation level	U[0,3]
Bandwidth resources	U[50,100]
VNRs	2000
VNR nodes	U[2,10]
Computing requirements	U[0,50]
Storage requirements	U[0,50]
Isolation requirements	U[0,3]
Bandwidth requirements	U[0,50]
VNRs for training	1000
VNRs for testing	1000

TABLE II
PARAMETERS OF THE DRL-BASED LEARNING AGENT

Parameters	Value
Learning rate (α)	0.005
Discount factor (γ)	0.998
Batch Size (<i>batch_size</i>)	100
Number of epoch (<i>numEpoch</i>)	100

E. Evaluation Results

The comparison algorithms are heuristic and do not take into account the long term impact of resources and isolation for allocating VNRs.

Figure 2 shows the acceptance ratio of all the algorithms. Because available SN resources are readily available, the acceptance ratio of all algorithms decreases at first in the beginning. However, as time passes, the SN nodes become more occupied, making it more difficult to accommodate more VNRs.

Figure 3 depicts the average revenue, which has a downward trend over time, similar to figure 2. This is due to the fact that both metrics are related to the amount of available resources in SN. On the other hand Figure 4, shows an upward trend because the long-term revenue to cost ratio is independent of available SN resources.

V. CONCLUSION

In this paper, we propose drl_vne algorithm and compare its performance with previous related works to solve the problem. In the future, we intend to explore our problem model and solution that supports 1) VNRs with changing resource requirements

TABLE III
EXISTING ALGORITHMS FOR COMPARISON

rw_rank [7]	Algorithm inspired from the Goolge PageRank algorithm
grc_rank [8]	Algorithm based on the global resource capacity.
mct_vne [9]	Algorithm based RL that uses Monte-Carlo Tree Search (MCT).
drl_vne (Our)	Based on DRL which considers isolation as an attribute during VNR embedding.

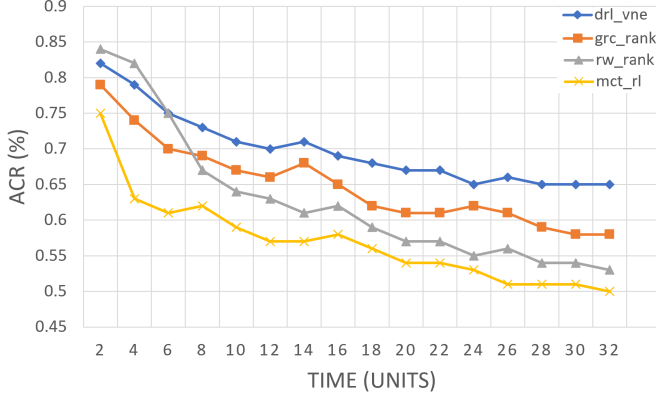


Fig. 2. Acceptance rate ratio

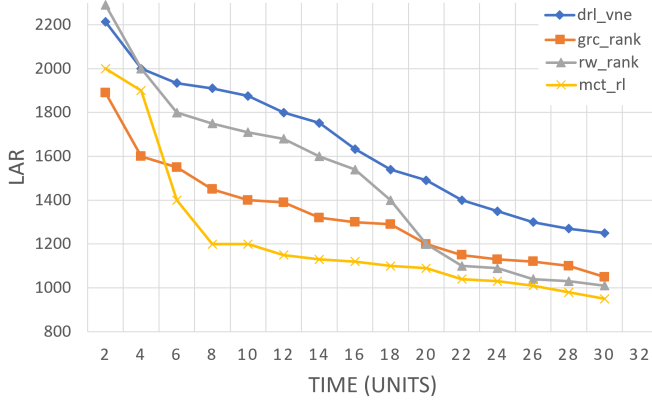


Fig. 3. Long-term average revenue

after allocation; 2) embedding one VNR node over multiple SN nodes; 3) comparison of our proposed algorithm with three more algorithms proposed in related works.

REFERENCES

- [1] A. Gohar and G. Nencioni, "The role of 5g technologies in a smart city: The case for intelligent transportation system," *Sustainability*, vol. 13, no. 9, 2021. [Online]. Available: <https://www.mdpi.com/2071-1050/13/9/5188>
- [2] H. Cao, S. Wu, Y. Hu, Y. Liu, and L. Yang, "A survey of embedding algorithm for virtual network embedding," *China Communications*, vol. 16, no. 12, pp. 1–33, 2019.
- [3] H. Cao, "Exact solutions of vne: A survey," *China Communications*, vol. 13, no. 6, pp. 48–62, 2016. [Online]. Available: <http://www.cic-chinacommunications.cn/EN/Y2016/V13/I6/48>

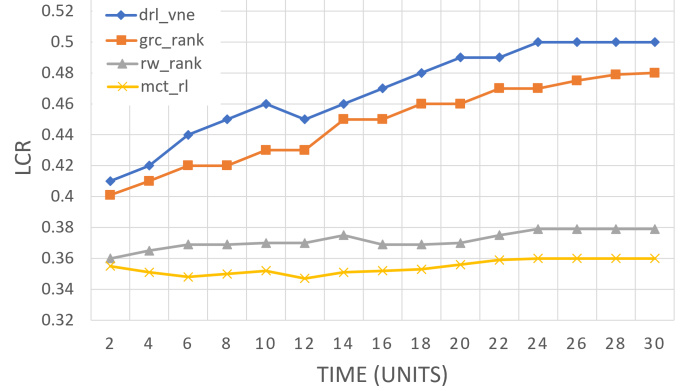


Fig. 4. Long-term average revenue to cost ratio

- [4] A. J. Gonzalez, J. Ordóñez-Lucena, B. E. Helvik, G. Nencioni, M. Xie, D. R. Lopez, and P. Grønsund, "The isolation concept in the 5g network slicing," in *2020 European Conference on Networks and Communications (EuCNC)*, 2020, pp. 12–16.
- [5] . OpenAI et al., "Dota 2 with large scale deep reinforcement learning," *ArXiv*, vol. abs/1912.06680, 2019. [Online]. Available: <https://doi.org/10.48550/arXiv.1912.06680>
- [6] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, "Emergent tool use from multi-agent autocurricula," *ArXiv*, vol. abs/1909.07528, 2020.
- [7] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, and J. Wang, "Virtual network embedding through topology-aware node ranking," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 2, p. 38–47, apr 2011. [Online]. Available: <https://doi.org/10.1145/1971162.1971168>
- [8] L. Gong, Y. Wen, Z. Zhu, and T. Lee, "Toward profit-seeking virtual network embedding algorithm via global resource capacity," in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, 2014, pp. 1–9. [Online]. Available: <https://doi.org/10.1109/INFOCOM.2014.6847918>
- [9] I. Fajjari, N. Aitsaadi, G. Pujolle, and H. Zimmermann, "Vne-ac: Virtual network embedding algorithm based on ant colony metaheuristic," in *2011 IEEE International Conference on Communications (ICC)*, 2011, pp. 1–6. [Online]. Available: <https://doi.org/10.1109/icc.2011.5963442>
- [10] S. Hougardy, "The floyd-warshall algorithm on graphs with negative cycles," *Information Processing Letters*, vol. 110, no. 8, pp. 279–281, 2010. [Online]. Available: <https://doi.org/10.1016/j.ipl.2010.02.001>
- [11] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. The MIT Press, 2020.
- [12] M. O'Sullivan, L. Aniello, and V. Sassone, "A methodology to select topology generators for WANET simulations (extended version)," *CoRR*, vol. abs/1908.09577, 2019. [Online]. Available: <http://arxiv.org/abs/1908.09577>
- [13] . Abadi et al., "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283. [Online]. Available: <https://doi.org/10.5555/3026877.3026899>
- [14] J. Guo and D. Kunin, "Ai notes: Initializing neural networks," 2019. [Online]. Available: <https://www.deeplearning.ai/ai-notes/initialization/index.html>