Automatic Virtual Network Embedding: A Deep Reinforcement Learning Approach With Graph Convolutional Networks

Zhongxia Yan, Jingguo Ge, Yulei Wu, Liangxiong Li, and Tong Li

Mohammad Khalaji

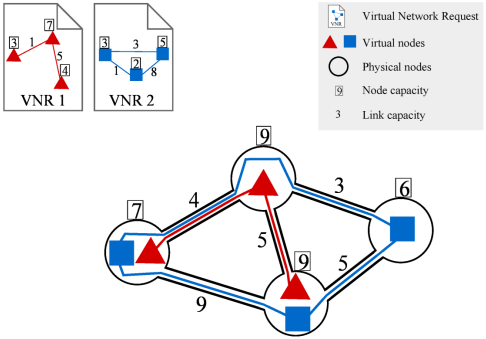
David R. Cheriton School of Computer Science, University of Waterloo

March 2022

Virtual Networks

- The internet structure is extremely rigid when it comes to architectural changes.
- It cannot accommodate the growing emergence of network services.
- Network virtualization (NV): An abstraction layer between the physical network and a virtual, software-based network.
- NV decouples network services from the underlying hardware.

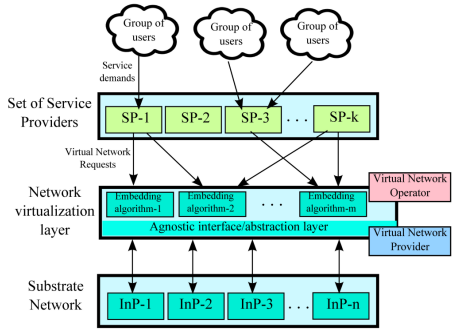
Virtual Network Embedding (VNE)



Minimal example [3]

- Dynamic mapping of virtual resources into physical hardware in the substrate network (SN)
- "Resource Allocation"
- Two subproblems:
  1. Virtual Node Mapping (VNoM)
  2. Virtual Link Mapping (VLiM)
- NP-Hard

Virtual Network Embedding (VNE)



VNE flow [3]

Reinforcement Learning

- At each time step t, the agent receives state s_t by observing the environment
- An action a_t is generated and performed
- The agent is moved onto next state s_{t+1}, and receives reward r_t
- The primary goal is to maximize the estimation of expected rewards:

$$\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$$

Substrate Network (SN)

Substrate Network

A substrate network is a weighted, undirected graph $G_s = (N_s, L_s, A^n, A^l)$

- N_s: The collection of substrate nodes
- L_s: The collection of substrate links
- A^n: Node attributes (e.g., CPU frequency, Memory)
- A^l: Link attributes (e.g., bandwidth, latency)

Virtual Network (VN)

Virtual Network

A virtual network is a weighted, undirected graph $G_v = (N_v, L_v, R^n, R^l)$

- N_v: The set of virtual nodes
- L_v: The set of virtual links
- R^n: Virtual node requests for substrate nodes
- R^l: Virtual link requests for substrate links

Virtual Network Request (VNR)

Virtual Network Request

A VNR can be denoted as $VNR = (G_v, t_a, t_d)$

- G_v: Virtual topology
- t_a: Arrival time
- t_d: Departure time

The VNE Problem

VNE

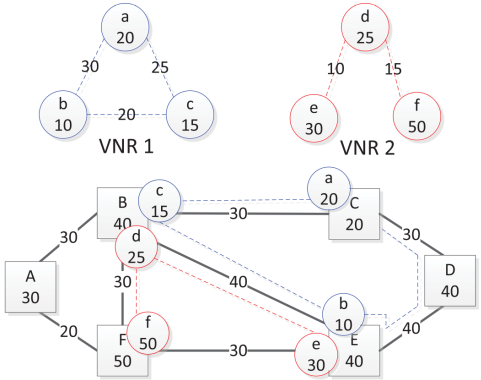
A VNE can be defined as a mapping: $M : G_v(N_v, L_v) \longrightarrow G_s^{'}(N_s^{'}, L_s^{'})$

- $G_s^{'}$: A subgraph of G_s
- $N_s^{'} \subset N_s$
- $L_s^{'} \subset L_s$

The mapping procedure is composed of two stages:

1. **Node mapping:** To host virtual nodes on substrate nodes with sufficient resources according to R^n
2. **Link mapping:** Assigning virtual links onto loop-free substrate paths, making sure that they satisfy the virtual link requested resources R^l

The VNE Problem



Optimization Objectives

1. **Acceptance Ratio:** What percentage of VNRs are being successfully embedded into the SN?
2. **Long-Term Average Revenue:** The more resources are demanded, the more profit made by infrastructure owned.

$$Rev(G_v) = \sum_{n_v \in N_v} CPU(n_v) + \sum_{l_v \in L_v} BW(l_v)$$

$$Rev(T) = \frac{1}{T} \sum_{t=0}^{T} Rev(G_v^t)$$

3. **Running Time:** How long does it take for a VNR to be successfully processed?

Other Work

VNE Algorithm	Description
D-ViNE [2]	Use a deterministic rounding-based approach to attain a linear programming relaxation of the mixed-integer programming (MIP) that corresponds to the VNE problem, aiming to minimize the cost of VNRs.
R-ViNE [2]	Same as D-ViNE, except for its rounding approach which is randomly decided.
NodeRank [1]	A node-ranking based algorithm that inspires from Google's PageRank.
GRC [4]	A node-ranking based algorithm that manages global resource capacity.
MCVNE [5]	A RL-based algorithm that uses Monte-Carlo Tree Search to search the action space.

Drawbacks of other work

- Manually determined constraints and features $\longrightarrow$ Less room for optimization
- Explicit and single-objective optimization targets $\longrightarrow$ Less flexible algorithms
  - Example: Focusing on reducing the cost
- Running time

RL Problem

The RL solution to the VNE problem must incorporate the typical components of reinforcement learning:

- States
- Actions
- Rewards

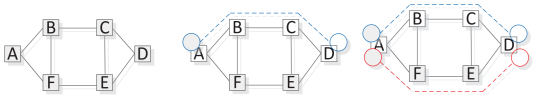
State Representation

State Representations	Description
S_CPU_Max	The maximum of the CPU resources over all SN nodes.
S_BW_Max	The max bandwidth of each substrate node. We define the bandwidth of a node as the sum of all links' bandwidth that directly link to this node.
S_CPU_Free	The amount of the CPU resources that are currently free on every substrate node.
S_BW_Free	The bandwidth resources that are yet to be allocated on all substrate nodes.
Current_Embedding	The (partial) embedding result of the current VNR. Each substrate node is set to 1 if it hosts a virtual node in the current VNR and 0 otherwise. This feature works as a mask to prevent virtual nodes in the same VNR from sharing one substrate node, as most previous works did.
V_CPU_Request	The number of virtual CPUs the current virtual node needs to fulfill its requirement.
V_BW_Request	The total bandwidth the current virtual node demands according to the current VNR.
Pending_V_Nodes	The number of unallocated virtual nodes in the current VNR.

Action Definition

- **Action:** A valid embedding process, allocating VNRs onto subsets of SNs
- **Naive Approach:** Every possible subgraph belongs to the action space
  - Grows exponentially as nodes and links are added
- **Computationally Consistent Approach:** The set of substrate nodes
- At every step of the execution of an action:
  1. Focus on exactly one virtual node, and generate a suitable substrate node to host it
  2. Perform a "Hybrid Search" to find appropriate substrate paths to host virtual links corresponding to the recently hosted virtual node

Hybrid Search



1. Try the shortest path
2. If fails, search in a set of edge-disjoint paths

Algorithm 1 Virtual Network Embedding Procedure

Input:
The VNR topology G_v;
The substrate network topology G_s;
The already embedded virtual node list l;
The substrate node list s that hosts nodes in l;
The currently processing virtual node n;
The selected action (i.e. substrate node) a.

Output:
The embedding result of action a.

if $S\_CPU\_Free[a] < V\_CPU\_Request[n]$ **then**
  return $ACTION\_FAILED$;
end if
Embed virtual node n onto substrate node a;
for $i = 0$; $i < length(l)$; $i++$ **do**
  if $(l[i], n) \in EdgeSet(G_v)$ **then**
    Find a substrate path p in G_s that links $s[i]$ and a following given search type: shortest-path, full or hybrid;
    for all substrate links e in path p **do**
      if $BW\_Free(e) < BW\_Request((l[i], n))$ **then**
        Undo all previous embedding actions and release all resource secured by the current VNR;
        return $ACTION\_FAILED$;
      end if
    end for
    Embed virtual link $(l[i], n)$ onto substrate path p;
  end if
end for
return $ACTION\_SUCCESSFUL$;

Reward

- Need to distinguish between 'slightly good' and 'really good' actions
- **Reward shaping:** Designing the reward function
  1. Acceptance ratio
  2. Cost efficiency
  3. Load balancing
  4. Exploration/exploitation

Acceptance Ratio

$$r_a = \begin{cases} 100\gamma_a & a_t \text{ is successful} \\ -100\gamma_a & \text{otherwise} \end{cases}$$

- γ_a is a discount factor that starts from $\frac{1}{|N_v|}$ and gradually increases to 1 when the last virtual node is being processed
- Latter nodes have less embedding options, so they deserve a greater weight

Cost efficiency

$$r_c = \frac{\delta(revenue)}{\delta(cost)}$$

- Better embedding policies consume less substrate resources
- Example: Virtual links hosted on short substrate paths

Load balancing

$$r_s = \frac{S\_CPU\_REMAINING[a]}{S\_CPU\_MAX[a]}$$

- The agent is encouraged to pick substrate nodes with more spare resources

Exploration/exploitation

- Eligibility trace

$$egb\_trace_t[i] = \begin{cases} \gamma_e(egb\_trace_{t-1}[i] + 1) & i == a_t \\ \gamma_e egb\_trace_{t-1}[i] & otherwise \end{cases}$$

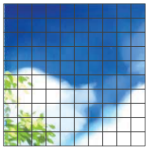
- γ_e is a decay factor
- Repetitive actions receive a high *egb_trace*
- Unpicked actions gradually decay to 0

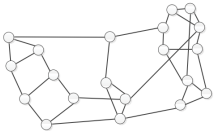
Final Reward

$$Reward[a_t] = \frac{r_a r_c r_s}{egb\_trace[a_t] + \epsilon}$$

Feature Extraction

- **First Approach:** Convolutional neural networks (CNNs)
- **Drawback:** CNNs work best in Euclidean domains





- **Alternative Approach:** Use the Laplacian matrix and orthogonal factorization to characterize the spatial features

Feature Extraction

- **GCN:** Graph Convolutional Network
- **Fourier Transform:** An n-dimensional vector can be represented as a set of orthogonal vectors
- Assume G has n nodes, and the features of nodes are gathered as a vector $x \in \mathbb{R}^n$
- Graph convolution:

$$(x * y)_G = U((U^T y) \odot (U^T x))$$

$$y_{out} = \sigma(\sum_{k=0}^{K} \alpha_k L^k x)$$

- **K:** The "hop" size
- 60 features per substrate node $\longrightarrow$ $60 \times |N_s|$ matrix of features

Policy Generation

Simple steps:

1. Transform the extracted features to a vector with $60 \times |N_s|$ values
2. Pass the vector through a fully-connected layer to get $|N_s|$ values
3. Use Softmax to be able to interpret the data as probabilities

Parallel Policy Gradient

- Asynchronous Advantage Actor-Critic (A3C)
  - Advantage function
  - Parallel learning algorithm based on a "master-worker" scheme
- Actor network (θ) generates policy π_θ
- Critic network (θ_v) generates value estimation $V^{\pi_\theta}(s_t; \theta_v)$

Advantage Function

- Traditional policy gradient:

$$\nabla_\theta \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t r_t] = \mathbb{E}_{\pi_\theta}[\nabla_\theta log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)]$$

- Advantage function:

$$A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) = r_t + \gamma V_\theta^{\pi}(s_{t+1}) - V^{\pi_\theta}(s)$$

- Advantage Actor-Critic (A2C):

$$\nabla_\theta \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t r_t] = \mathbb{E}_{\pi_\theta}[\nabla_\theta log \pi_\theta(s,a) A^{\pi_\theta}(s,a)]$$

Updating the Networks

Actor

$$\theta \longleftarrow \theta + \alpha \sum_{t} \nabla_\theta log \pi_\theta(s_t, a_t) A^{\pi_\theta}(s_t, a_t) + \beta \nabla_\theta H(\pi_\theta(.|s_t))$$

- α: Learning rate
- $\pi(.|s_t)$: Policy over all actions while in s_t
- H: The entropy of the policy; Acts as a regularization agent encouraging exploration
- β: Decaying exploration parameter

Updating the Networks

Critic

$$\theta_v \longleftarrow \theta_v + \alpha^{'} \sum_{t} \nabla_{\theta_v} (r_t + \gamma V_{\theta}^{\pi}(s_{t+1}; \theta_v) - V_{\theta}^{\pi}(s_t; \theta_v))^2$$

- $\alpha^{'}$: Learning rate
- $V_{\theta}^{\pi}(.; \theta_v)$: Value function estimation

Training Challenges

- Two main challenges:
  1. Gathering experience from the environment is slow
  2. The $\{s_t, a_t\}$ pairs in a trajectory are highly correlated
- **Solution:** Parallel training
  - Use 24 individual agents to collect trajectories
  - On 24 *independent* copies of the network

Master

Algorithm 2 Parallel Training Algorithm - Master

Initialize the *actor network* and *critic network*;
Initialize the number of workers $NUM\_WORKERS$;
for i in $NUM\_WORKERS$ **do**
    Create a *worker* agent $w[i]$ with a same copy of *actor network* and *critic network*;
end for
while TRUE **do**
    for i in $NUM\_WORKERS$ **do**
        Collect the experiences generated by $worker[i]$;
    end for
    Adjust the parameters in both *actor network* and *critic network* using previously collected experiences under policy gradient training method;
    for i in $NUM\_WORKERS$ **do**
        Push the newest version of *actor network* and *critic network* to $w[i]$;
    end for
end while

Worker

Algorithm 3 Parallel Training Algorithm - Worker

1: Initialize the *actor network* and *critic network*;
2: Initialize the independent *environment* for VNE;
3: **while** TRUE **do**
4: Receive the parameters of the *actor network* and *critic network* from *master*;
5: Sample a trajectory from the *environment* using the copied network;
6: Send the trajectory as $\{s_t, a_t, r_t, s_{t+1}\}$ experience tuples to *master*;
7: **end while**

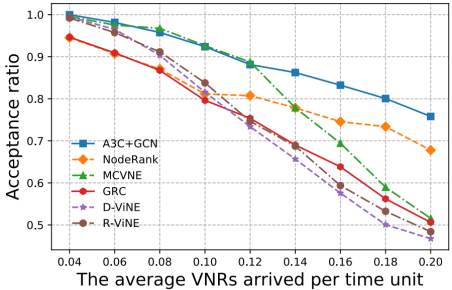
Evaluation Settings

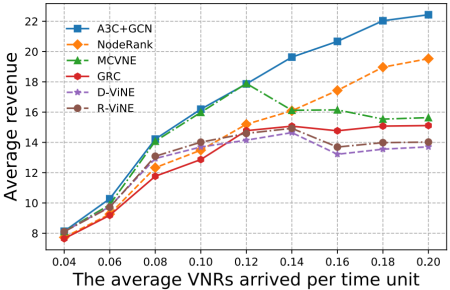
- Random substrate network topology (the Waxman random graph)
- Substrate network: 100 nodes and 500 links
- CPU and bandwidth: uniformly distributed between 50 and 100
- VNR arrival rate: 4 per 100 time units
- VNR resource requests: uniformly distributed from 0 to 30
- The testing phase: 50,000 time units, so there are 2,000 VNRs

Comparable Algorithms

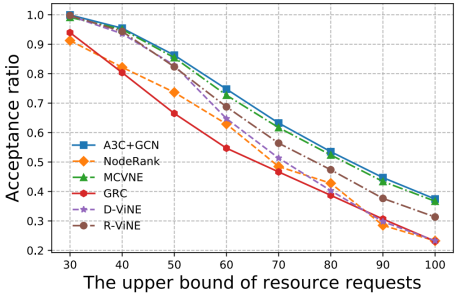
VNE Algorithm	Description
D-ViNE [2]	Use a deterministic rounding-based approach to attain a linear programming relaxation of the mixed-integer programming (MIP) that corresponds to the VNE problem, aiming to minimize the cost of VNRs.
R-ViNE [2]	Same as D-ViNE, except for its rounding approach which is randomly decided.
NodeRank [1]	A node-ranking based algorithm that inspires from Google's PageRank.
GRC [4]	A node-ranking based algorithm that manages global resource capacity.
MCVNE [5]	A RL-based algorithm that uses Monte-Carlo Tree Search to search the action space.

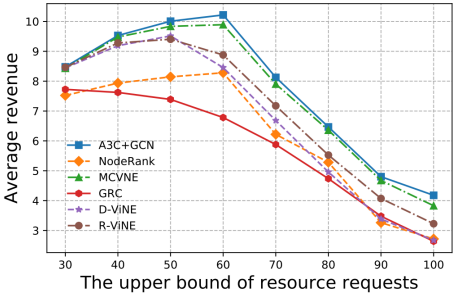
Arrival Rate



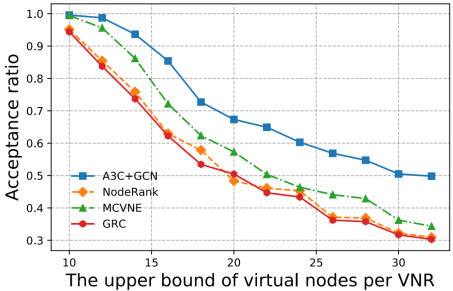


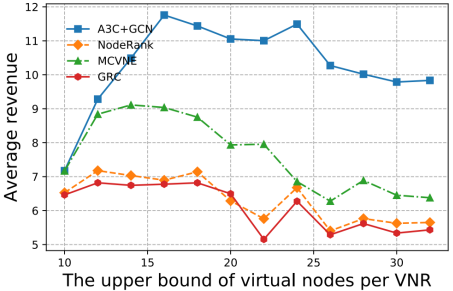
Resource Request





Node Size Expansion





Running Time

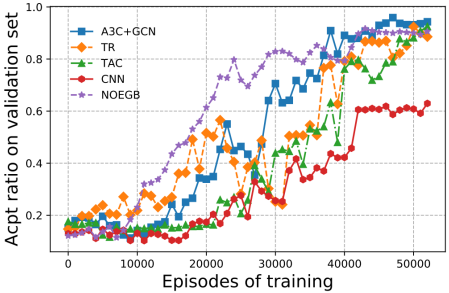
	Arrival Rate Tests	Resource Request Tests	Node Size Expansion Tests
A3C+GCN	0.219	0.227	0.476
NodeRank	0.103	0.125	0.388
GRC	0.086	0.079	0.135
MCVNE	1.815	1.893	9.649
D-ViNE	23.778	21.488	-*
R-ViNE	22.181	21.925	-*

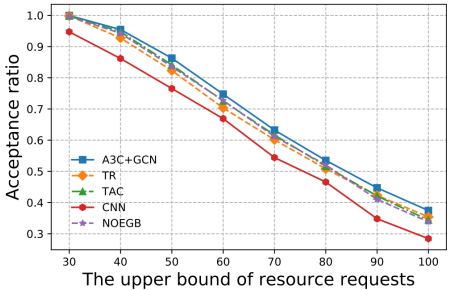
* Runs in excess of computing resource and unable to get a result.

Validation

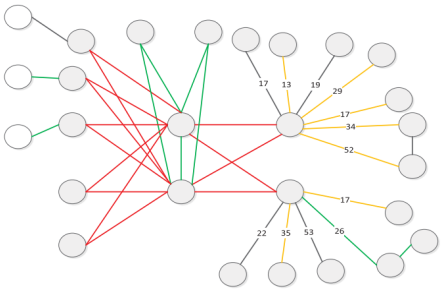
VNE Algorithm	Description
TR	Traditional reward function
TAC	A3C with one worker agent
CNN	Traditional CNN feature extraction
NOEGB	No eligibility trace in the reward function

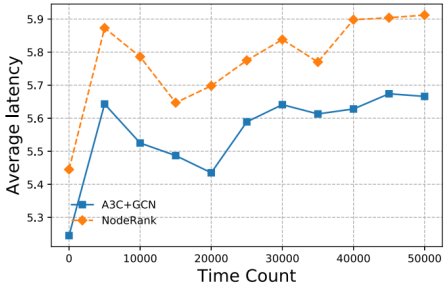
Validation


A3C+GCN
TR
TAC
CNN
NOEGB
Acpt ratio on validation set
Episodes of training
1.0
0.8
0.6
0.4
0.2
0
10000
20000
30000
40000
50000

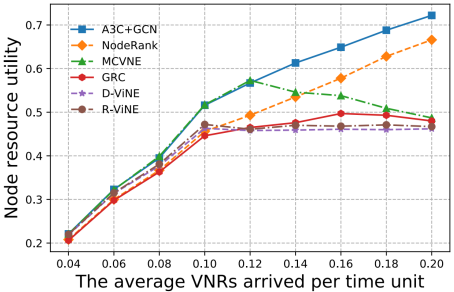

A3C+GCN
TR
TAC
CNN
NOEGB
Acceptance ratio
The upper bound of resource requests
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
30
40
50
60
70
80
90
100

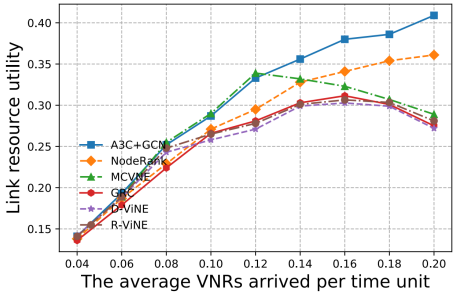
Feasibility Study





Feasibility Study





Main Contributions

- Automatic VNE based on deep reinforcement learning
- Parallel policy gradient training algorithm (A3C)
- 3-Order GCN instead of traditional CNN
- Multi-objective reward function
- Up to %39.6 and %70.6 improvement in two of the objectives

Room for Improvement

- Train the model on a larger SN topology
- Simulate larger, more demanding VNRs
- Dynamically changing VNRs even after embedding
- Simulating node and/or link failures over the SN
- One-to-many mapping of virtual nodes

References

[1] Xiang Cheng et al. "Virtual network embedding through topology-aware node ranking". In: *ACM SIGCOMM Computer Communication Review* 41.2 (2011), pp. 38–47.

[2] Mosharaf Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. "Vineyard: Virtual network embedding algorithms with coordinated node and link mapping". In: *IEEE/ACM Transactions on networking* 20.1 (2011), pp. 206–219.

[3] Andreas Fischer et al. "Virtual network embedding: A survey". In: *IEEE Communications Surveys & Tutorials* 15.4 (2013), pp. 1888–1906.

[4] Long Gong et al. "Toward profit-seeking virtual network embedding algorithm via global resource capacity". In: *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE. 2014, pp. 1–9.

[5] Soroush Haeri and Ljiljana Trajković. "Virtual network embedding via Monte Carlo tree search". In: *IEEE transactions on cybernetics* 48.2 (2017), pp. 510–521.