An Intelligent Parallel Algorithm for Online Virtual Network Embedding

Khoa TD Nguyen[†] and Changcheng Huang[†]
[†]Department of Systems and Computer Engineering
Carleton University, Ottawa, ON K1S 5B6, Canada
{*khoatnguyen, huang*}@*sce.carleton.ca*

***Abstract*—Network virtualization is ubiquitously an essential attribute to enable the success of the future virtualized networks (e.g. forthcoming 5G network, smart Internet of Things (IoT)). Virtual Network Embedding (VNE) is the main challenge in network virtualization that allows multiple heterogeneous Virtual Networks (VNs) to simultaneously coexist on top of a shared substrate infrastructure. Many VNE algorithms have been proposed over past decades but most of them are merely focusing on VNE node mapping and leaving link mapping task for the popular k-shortest path algorithms or multi-commodity flow (MCF) mechanism. In this paper, we propose an intelligent VNE orchestration for link mapping stage which exploits distributed parallelism to considerably reduce the processing time with high efficiency. Extensive simulations have shown that our proposed algorithm outperforms the most popular VNE algorithms.**

***Index Terms*—Virtual Network Embedding, Parallel Algorithm, 5G network, IoT, Artificial Intelligent.**

I. Introduction

In recent years, network virtualization (VN) has been receiving significant attention from both industry and academia as it is a vast promising paradigm for the success of the next generation networks such as virtualized 5G network [1], IoT virtualized networks [2]. NV enables sharing the underlying substrate resources among multiple virtual networks seamlessly and enabling isolated coexistence of multiple VNs on a single substrate network (SN), which prevents the infrastructure expansion and improves network utilization. Additionally, lower hardware costs for computing have benefited parallel algorithms in dealing with complex computing tasks. In general, the service provider (SP) converts an application or a service into a VN and then deliver to an infrastructure provider (InP) as a request. InP tries to map the corresponding VN with a set of nodes connected via links to make up a specific topology onto its infrastructure through an optimization process with multiple constraints. In fact, VN requests (VNRs) dynamically arrive and stay in the network during a random duration in most real-life scenarios. Due to its intricacy, scalability and time consumption to achieve optimal solutions within polynomial time, the formulated optimization models such as Integer Linear Programming (ILP) are not tailored for online VNE problems.

Conceptually, VNE process that is to embed requested VNs onto a underlying shared SN can be divided into two sub-problems: Virtual Node Mapping (VNoM) and Virtual Link Mapping (VLiM). It is worth noting that VLiM is more challenging than the counterpart VNoM because of its special requirements since all substrate links that a constituent path traverses for the embedding solutions of a requested virtual link must necessarily have enough residual capacities, which causes bandwidth fragmentation more likely to occur. Embedding the virtual links on the underlying shared SN under stringent constraints is still NP-Hard [3]. Furthermore, most research work [3], [4], [5], [6], [7], [8], [9] focuses on node mapping and leaves link mapping stage for only k-shortest path or multi-commodity flow (MCF) algorithms, which obviously restricts VNE link mapping options.

Along with 5G virtualized networking and IoT, embedding VNs where the physical network enables path splittable and unsplittable configurations is evidently also a fundamental research aspect in Software Defined Network (SDN), Network Function Virtulization (NFV) and Future Edge Clouds. Though splittable-based embedding obtains better resource utilization in theory, it was declared to be simpler than the unsplittable mechanism and to generate a larger overhead to consistently maintain the network state [10]. Furthermore, such mechanism may result in out-of-order package delivery that may introduce unacceptable latency for delay-sentitive applications. In this paper, we present an intelligent algorithm that exploits distributed parallel machines dealing with VNE link mapping problems to achieve near-optimal solutions and considerably cut down the operation time.

The remainder of this paper is organized as follows: The related work is presented in Section II. Section III formulates the network model and then we introduce the parallel genetic algorithm for VNE link mapping in Section IV. We present the performance evaluation of the proposed algorithm in Section V. Section VI is a conclusion of this paper.

II. Related Work

With the demanding research efforts made to network virtualization, [11] has provided a comprehensive survey to this research field. [12] resolved the one-by-one online embedding scheme where virtual network requests dynamically arrive and depart. This mapping problem is conclusively proven NP-hard. A path splittable-support mapping of a virtual link over multiple substrate paths, which deposes the link embedding issue to multicommodity flow problem by considering a virtual link as a commodity is introduced in [3]. In contrast, a coordination approach between node and link mapping solving NP-hard node embedding problem that relaxes the integer constraints to gain a linear program and then benefits rounding techniques to select distinct node mapping is first proposed in [4]. Nevertheless, the paper that extends [4] by efficiently supporting node splitting schemes and node collocation is essentially proposed in [5]. Meanwhile, Genetic Algorithms (GA), one of the approaches widely used in Artificial Intelligence (AI), applied to VNE problems were first studied in [6] and [7]. Node ranking methods based on GA algorithms with topology attribute concerns were also proposed in [6]. Research work in [7] eventually makes a performance comparison among Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO) and

GA in which they were concerned with node mapping stage. A VNE model based on GA algorithm that handles multiple InP domains is investigated in [8] while a research study [9] revised the typical GA algorithm by reordering the mutation phase to generate higher quality offspring in initial population.

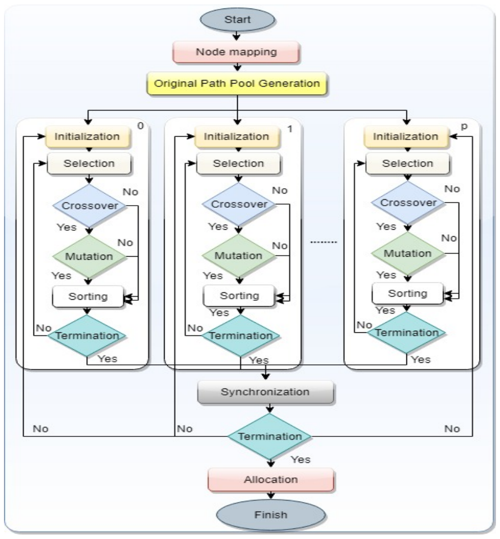


Fig. 1: *Parallel operation scheme*

III. Network Model and Problem Descriptions

A. Virtual Network Assignment

A substrate network is modelled as a weighted undirected graph and denoted as $G^s = (N^s, L^s)$ where N^s is the set of substrate nodes and L^s is the set of substrate links. Each substrate node $n^s \in N^s$ that has a geographic location $loc(n^s)$ is associated with the available CPU capacity weight value $C(n^s)$, while each substrate link $l^s \in L^s$ between two substrate nodes has the bandwidth capacity value $B(l^s)$. For simplification, memory and storage resources are not considered in this paper. Similarly, we model the i^{th} arriving VNR as a weighted undirected graph denoting by $G_i^v = (N_i^v, L_i^v)$, where N_i^v and L_i^v are the set of virtual nodes and virtual links of the i^{th} VNR respectively. Each virtual node $n_i^v \in N_i^v$ is associated with a CPU requirement $c(n_i^v)$ and the location preference $loc(n_i^v)$, while each virtual edge $l_i^v(s^i, d^i) \in L_i^v$ between the corresponding virtual source s^i and destination nodes d^i has a bandwidth requirement $b(l_i^v)$. Embedding a VNR G_i^v onto the corresponding G^s can be dissolved into two major components as discussed above: Virtual Node Mapping (VNoM) and Virtual Link Mapping (VLiM). In node mapping, each virtual node from the same VNR can be mapped to a substrate node $\mathcal{A_N} : N_i^v \rightarrow N^s$ with $n^v \in N_i^v$ subject to:

$$C(n^v) \leq R_N(\mathcal{A_N}(n_i^v)) \tag{1}$$

$$D(loc(n_i^v), loc(\mathcal{A_N}(n_i^v))) \leq loc(n_i^v) \tag{2}$$

$$\mathcal{A_N}(n_i^v) \in N^s \tag{3}$$

$$R_N(n^s) = C(n^s) - \sum_{n^v \rightarrow n^s} C(n_i^v) \tag{4}$$

where $i \rightarrow j$ denotes the virtual node i is hosted on the substrate node j, $D(i,j)$ measures the distance between the locations of node i and j and $R_N(n^s)$ is the residual or the available CPU capacity of a substrate node. In contrast, each virtual link can be mapped to a substrate path or a defined set of substrate paths between the corresponding substrate nodes hosting the virtual nodes of such virtual link. It is defined by $\mathcal{A_L} : L_i^v \rightarrow L^s$ with $l^v = (s^v, t^v) \in L_i^v$, $\mathcal{E}^s$ is a set of substrate paths from a source node to a destination node.

$$\mathcal{A_L}(s^v, t^v) \subseteq \mathcal{E}^s(\mathcal{A_N}(s^v), \mathcal{A_N}(t^v)) \tag{5}$$

subject to:

$$\sum_{E \in \mathcal{A_L}(l^v)} R_L(E) \geq B(l^v) \tag{6}$$

$$R_L(E) = min_{l^s \in E} R_L(l^s) \tag{7}$$

$$R_L(l^s) = B(l^s) - \sum_{l^v \rightarrow l^s} B(l^v) \tag{8}$$

where $R_L(E)$ and $R_L(l^s)$ are the available bandwidth of a substrate path $E \in \mathcal{E}^s$ and the residual capacity of a substrate edge respectively.

B. Performance metrics

In this paper, the revenue is defined as the sum of total virtual resources embedded to the SN over time. The revenue for the i^th VNR G_i^v is defined below:

$$R(G_i^v) = w_\alpha * \sum_{l_i^v \in L_i^v} b(l_i^v) + w_\beta * \sum_{n_i^v \in N_i^v} c(n_i^v) \tag{9}$$

where $b(l_i^v)$ and $c(n_i^v)$ are the bandwidth requirement of the virtual link l_i^v and the CPU requirement of the virtual node n_i^v while w_α and w_β are the unit weights of the embedded bandwidth and CPU resources respectively.

Acceptance ratio: the ratio between the number of accepted virtual network requests over the number of virtual network requests arriving in the time interval τ is calculate as below:

$$A_c^\tau = \left| \frac{\chi^a(\tau)}{\chi(\tau)} \right| \tag{10}$$

where $\chi^a(\tau)$ and $\chi(\tau)$ denotes the number of the successfully embedded VNRs and the number of virtual network requests.

Cost: similarly, we define the cost of the i^{th} VNE $C(G_i^v)$ as the sum of total substrate resources allocated to the i^{th} virtual network.

$$C(G_i^v) = \sum_{n_i^v \in N_i^v} c(n_i^v) + \sum_{l_i^v \in L_i^v} \sum_{l^s \in L^s} f_{l^s}^{l_i^v} \tag{11}$$

where $f_{l^s}^{l_i^v}$ is the bandwidth of substrate link l^s that is allocated for the virtual link l_i^v

Fitness Function (FF): This is used to evaluate the quality of each of VLiM solutions among several feasible ones, and it can also provision a scientific proof for electing the corresponding chromosomes in GA's operators. We add hop-count as an important factor into FF besides bandwidth since it is believed that minimizing hop-count would help the cost of a VNR minimized. Fitness function $\mathcal{F}(\mathcal{C}_i)$ is calculated as below:

$$\mathcal{F}(\mathcal{C}_i) = \sum_{l^s \in L^s} \left(\frac{b(l_i^v) * \alpha}{b(l_i^v) + R_L(l^s)} + \frac{1}{h_{l^s}} * \beta \right) \tag{12}$$

where, $\mathcal{C}_i$, h_{l^s} are a feasible solution and hop-count of l^s respectively. α and β are weight parameters equivalent to bandwidth or hop-count factors.

IV. Intelligent Distributed Parallel Resource-Allocation Algorithm

Recently, parallel computing is emerged as an effective solution to solve larger problems with time saving and low cost by providing concurrency. Moreover, adopted the idea of natural selection, GA is an intriguing AI approach for solving both constrained and unconstrained optimization problems. [13] also proved that the Genetic Algorithm can be naturally described as a parallel search and there is no dependency among different feasible solutions since they are mutually exclusive. Hence, we propose an intelligent parallel algorithm based on GA that can be running on a predefined number of distributed machines in which they are independently operating to generate feasible solutions denoted as chromosomes. Our proposed parallel GA scheme is present in Fig 1. As shown, we defined the procedures, which are functioning sequentially as working under a single master node such as node mapping, synchronization, etc,. while the ones independently operating parallel GA algorithms to find feasible solutions for VNE link mapping are working as several slave nodes. Each slave machine is independently running with defined iterations, the best feasible VLiM solution will be selected among parallel machines. Unlike previous papers that sequentially map requested virtual links, our proposed algorithm enables to map multiple link requests at once. A chromosome $\mathcal{C}_i$ is associated with a feasible link mapping solution of a VN. Whilst a gene g_i^j indicates a substrate path where i and j denote its current chromosome and virtual link respectively.

A. *Initial path pool generation*

Prior to link mapping procedures, we necessarily establish potential path database for mapping virtual links. For each pair of source-destination, a k-shortest path algorithm e.g. Dijkstras algorithm is simply implemented to determine k-shortest paths for the path pool generation. This primal process can be absolutely prepared prior online VNR arrivals.

B. *Slave node*

Population Initialization: each slave machine starts to operate GA algorithm with a population initialization step. Each chromosome $\mathcal{C}_i$ represents a feasible solution. Assume that there are N genes and M chromosomes, an initial population $\mathcal{P}$ (MxN size) at the machine k^{th} can be described as below:

$$\mathcal{P} = \begin{bmatrix} \mathcal{C}_1 \\ \mathcal{C}_2 \\ \vdots \\ \mathcal{C}_i \\ \vdots \\ \mathcal{C}_M \end{bmatrix} = \begin{bmatrix} g_1^1 & \cdots & g_1^j & \cdots & g_1^N \\ g_2^1 & \cdots & g_2^j & \cdots & g_2^N \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ g_i^1 & \cdots & g_i^j & \cdots & g_i^N \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ g_M^1 & \cdots & g_M^j & \cdots & g_M^N \end{bmatrix} \tag{13}$$

To form a chromosome, each gene which is associated with the mapping solution of a requested virtual link request must be uniformly selected from the initial path pool in random, which must pass a feasibility check to become a potential solution. This checking process is to ensure that the SN still has enough remaining resources to support such request. All N potential genes passed the feasibility process constitutes a chromosome, it is considered as a feasible solution for such corresponding VLiM.

Selection: selects the chromosome individuals as parents for the crossover operation. One or several pairs of parent chromosomes can be generally chosen from this step. Aimed at enhancing the degree of parallelism, we select the parents randomly with replacement from the initial population. However, the children produced in crossover may have either better or worse quality than their parents. Conceptually, we use fitness-based proportionate selection to select parents from the initial population, which relies on the cumulative sum of the fitness relative weights (12).

$$\mathcal{P} = \begin{bmatrix} \mathcal{C}_1 \\ \vdots \\ \mathcal{C}_s \\ \mathcal{C}_r \\ \vdots \\ \mathcal{C}_M \\ \mathcal{C}_{M+1} \\ \mathcal{C}_{M+2} \end{bmatrix} = \begin{bmatrix} g_1^1 & \cdots & g_1^{j^c} & g_1^{j^c+1} & \cdots & g_1^N \\ \vdots & \ddots & \vdots & \ddots & \vdots & \\ g_s^1 & \cdots & g_s^{j^c} & g_s^{j^c+1} & \cdots & g_s^N \\ g_r^1 & \cdots & g_r^{j^c} & g_r^{j^c+1} & \cdots & g_r^N \\ \vdots & \ddots & \vdots & \ddots & \vdots & \\ g_M^1 & \cdots & g_M^{j^c} & g_M^{j^c+1} & \cdots & g_M^N \\ g_s^1 & \cdots & g_s^{j^c} & g_r^{j^c+1} & \cdots & g_r^N \\ g_r^1 & \cdots & g_r^{j^c} & g_s^{j^c+1} & \cdots & g_s^N \end{bmatrix} \tag{14}$$

Crossover: combines two parent chromosomes to form new offspring of the next generation. We denote $\mathcal{C}_s$ and $\mathcal{C}_r$ as two parent chromosomes with their indexes s and r in initial population. Moreover, assume j^c is the random crossover point between any genes inbound the N length, and new descendant chromosomes are denoted as $\mathcal{C}_{(M+1)}$ and $\mathcal{C}_{(M+2)}$ respectively. The offsprings are produced by swapping two parent genes starting from the crossover point $j+1$ to the end as depicted in 14. Obviously, the quality of generated children can be worse than whose ancestors or the duplication of solutions may happen at different parallel levels. Crossover point j^c is randomly chosen, but should not start at the first or last gene in the parents chromosomes because the mated children are apparently same as their parents.

Mutation: applies random changes to individual parents to form new offspring. Mutation operation then includes randomly generating the mutation point denoted as j^m , and at that point, a new gene can replace the existing one of the in-processed chromosome to generate a new child. New selected gene which has been chosen from the original path pool must pass a feasibility check. Assume j^m is denoted as mutation point while $g_{r'}^{j^m}$ is a new gene that replaced the existing one in $\mathcal{C}_{(M+1)}$. The mutation solution $\mathcal{C'}_{(M+1)}$ after such substitution is $\mathcal{C'}_{(M+1)} = [g_s^1 \cdots g_{r'}^{j^m} \cdots g_s^N]$.

C. *Solution Sorting and Terminations*

The process running at each slave node is terminated when it reaches a predetermined number of iterations, and then the best solution among feasible ones after sorting based on its FFs delivers up to the next component called synchronization for global ranking. Generally, a parallel computation is constituted a series of ordered processes, waiting each other accomplished its all particular tasks is vulnerable to unexpected situations in which a process excessively takes long time to finish, that obviously affects total execution time, or two or more processes are jammed and waiting for each other to complete their work (e.g. deadlock). Thus, we decide the master GA procedure will be terminated when the best solution for VLiM has not been successively changed in t times, and t is defined as a termination parameter.

D. *Synchronization and VNR allocation*

After receiving all feasible VNR chromosomes from slave nodes, they are substantially ranked following their FF values to elect the final solution for the corresponding VN link mapping. As the result, VNR will be accepted and allocated into SN based on node and link mapping solutions. Then, the SN goes to update its residual resources.

E. *Execution Time Analysis*

Though there are elements of uncertainty which can increase the complexity of time calculation, a parallel and distributed computing model is expected to substantially reduce the

TABLE I: *Compared Algorithms*

Notation	Description
IDPA	Intelligent Distributed Parallel Algorithm
SP	Greedy Node Mapping with Shortest Path Based Link Mapping
R-ViNE	Random Node Mapping with Shortest Path Based Link Mapping
D-ViNE	Deterministic Node Mapping with Shortest Path Based Link Mapping

execution time of our proposed algorithms. We define $\mathcal{O}_m^i$, $\mathcal{O}_n^i$, $\mathcal{O}_o^i$, $\mathcal{O}_s^i$ and $\mathcal{O}_a^i$ as the execution time of master node, node mapping, original path pool generator, synchronization and VNR allocation respectively whilst i^{th} identifies the current VNR's index. We denote the parallel operation time of slave nodes as $\mathcal{O}_g^i$, that is depended on the last slave node which finishes its work, and the parallel level p as a trade-off between available substrate resource and the completion time. If the execution time of a sub-salve node is $\mathcal{X}_t$ in which t^{th} is slave node index $t \in [1, p]$, $\mathcal{O}_m^i$ and $\mathcal{O}_g^i$ are calculated as below:

$$\mathcal{O}_g^i = max\{\mathcal{X}_t\},\ t \in [1, p] \tag{15}$$

$$\mathcal{O}_m^i = \mathcal{O}_n^i + \mathcal{O}_o^i + \mathcal{O}_g^i + \mathcal{O}_s^i + \mathcal{O}_a^i \tag{16}$$

Similarly, if we denote $\mathcal{S}_m^i$ and $\mathcal{T}_g^i$ as the sequential operation time of master node and slave nodes respectively, they can be calculated as:

$$\mathcal{T}_g^i = \sum_{t=1}^{p} \mathcal{X}_t \tag{17}$$

$$\mathcal{S}_m^i = \mathcal{O}_n^i + \mathcal{O}_o^i + \mathcal{T}_g^i + \mathcal{O}_s^i + \mathcal{O}_a^i \tag{18}$$

Additionally, each slave node is independently operated in parallel with the same probability distribution and all nodes are mutually independent, so their execution time can be considered as independent identically-distributed random variables (iid) since the VNRs accordingly arrive in random. Following the numerical results in Section V, $\mathcal{X}_t$ obeys Normal Inverse Gaussian (NIG) as depicted in Fig 3b. We take advantage of the Chernoff-Cramer method that exploits the Moment Generating Function (MGF) to extract the upper tail bound of $\mathcal{O}_g^i$. As the result, MGF of $\mathcal{X}_t$ is illustrated as:

$$M_{\mathcal{X}}(z) = \mathbb{E}[e^{z*\mathcal{X}}] = e^{\left[z*\mu+\delta(\sqrt{\alpha^2-\beta^2}-\sqrt{\alpha^2-(\beta+z)^2})\right]} \tag{19}$$

with $\mu, \delta, \alpha, \beta > 0$ From Jensen's inequality equation [14], we have:

$$\begin{aligned} e^{z*\mathbb{E}[\mathcal{O}_g^i]} &\le \mathbb{E}[e^{z*\mathcal{O}_g^i}] = \mathbb{E}[max_t\{e^{z*\mathcal{O}_g^i}\}] \\ &= \sum_{t=1}^{p} \mathbb{E}[e^{z*\mathcal{X}_t}] = p\mathbb{E}[e^{z*\mathcal{X}_t}] \end{aligned} \tag{20}$$

Take a log of both sides, we have:

$$\begin{aligned} \mathbb{E}[\mathcal{O}_g^i] &\le log(p)\left[\mu + \frac{\delta}{z}(\sqrt{\alpha^2-\beta^2} - \sqrt{\alpha^2-(\beta+z)^2})\right] \\ &= log(p)\left[\mu + \frac{\delta\sqrt{\alpha^2-\beta^2}}{z}(1-\sqrt{1-\frac{z^2+2\beta z}{\alpha^2-\beta^2}})\right] \end{aligned} \tag{21}$$

$$\mathbb{E}[\mathcal{T}_g^i] = p\mathbb{E}[\mathcal{X}_t] \tag{22}$$

if $z, \mu, \alpha, \beta, \delta$ are constant values, we can see that the general tendency of parallel running is increasingly logarithmic, which is compared with the linear tendency of sequential operating in (20), it is concluded that our proposed algorithm considerably saves more time since increasing the parallel level p. If there are K number of VNRs and the total operation time of mapping all VNRs is denoted as $\mathbb{O}_g$, it can be calculated as below:

$$\mathbb{O}_g = \sum_{i=1}^{K} \mathcal{O}_m^i \tag{23}$$

V. Performance Evaluation

A. Simulation setup

We have implemented a discrete event simulator to evaluate the proposed algorithm with same settings as [4]. The substrate network and VN topologies are generated using GT-ITM which is a well-known topology generator. The SN is configured with 50 nodes, that are randomly placed on a 25×25 Cartesian plane, with 141 edges randomly connected using Waxman model with $\alpha = 0.5$ and $\beta = 0.2$. The value of α disposes of the maximal edge probability while β identifies the length of edges. The CPU and bandwidth resources of the SNs are uniformly generated between 50 and 100. VNRs arrive in Poisson process with an average rate of 4 to 8 VNs per 100 time units; and the VNR lifetimes follow an exponential distribution with an average of $\mu = 1000$ time units. In each VN graph, the number of virtual nodes is randomly determined by a uniform distribution between 2 and 10 with average VN connectivity at 50%. The CPU requirements of the virtual nodes are real numbers uniformly distributed between 0 to 20 randomly located on 25×25 grids while the bandwidth requirements of the virtual links are uniformly distributed between 0 to 50. Each simulation was running for $50,000$ time units that is longer than 50 times compared to the average lifetime of a virtual network. This time is enough long to achieve the sufficient number of independent samples. We plotted the graphs based on the average values with 95% confidence interval.

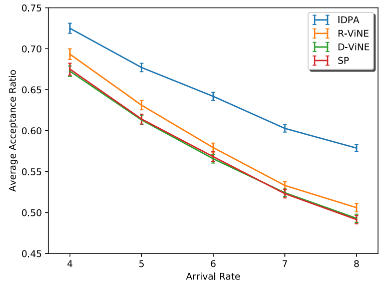
B. Compared Methods

In our evaluation, we compare our proposed GA algorithm with the rivals as described in Table I. They are vitally targeted because the VLiM algorithms that are performed well in [4] have been utilized in many VNE research papers. In detail, SP is simple and fastest algorithm which is considered as one of the most popular VN link mapping algorithm. Furthermore, we choose R-ViNE and D-ViNE algorithms since they are regularly provide the best performance by applying linear programming approach for node mapping.

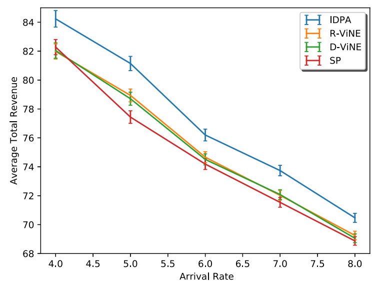
C. Evaluation Results

1) Execution Time Analysis: To study execution time distribution, we essentially collect manifold operation time of the parallel procedures at several slave nodes. Following the $\mathcal{X}_t$ histogram of IDPA as shown in Fig.3b, $\mathcal{X}_t$ basically fits Normal Inverse Gaussian distribution (NIG) that is defined as a normal variance-mean mixture where the mixing density is the inverse Gaussian distribution. The sum of square error of its fitting measured is $9.32e^{-3}$. The sample mean and standard derivation values of $\mathcal{X}_t$ of IDPA are 2.0363ms and 1.478 respectively. Moreover, the average total execution time of different algorithms is demonstrated in Fig.3c in which the parallel level p of IDPA algorithm is set to 16. We determine the $p = 16$ since we recognized that the overall performance of IDPA after numerous experiments achieves convergence.

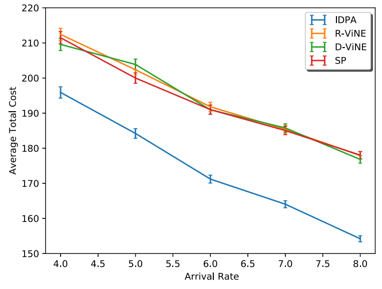
2) Performance Results: As delineated in Fig.2, IDPA algorithm accepts more VNRs with less cost than the rivals, which results in higher revenue for our algorithm. The dominant achievements are because our proposed approach efficiently



(a)

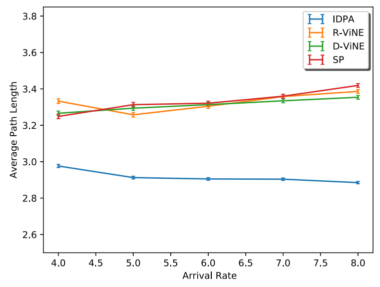


(b)

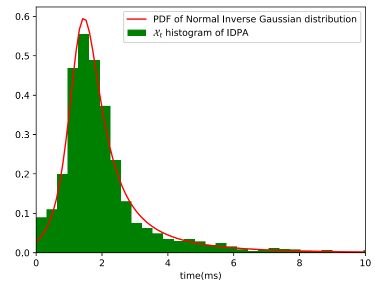


(c)

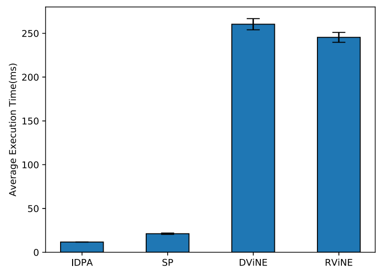
Fig. 2: **(a)** VNR Acceptance Ratio **(b)** Average generated avenue **(c)** Average cost of accepting VNR



(a)



(b)



(c)

Fig. 3: **(a)** Average path length **(b)** Time distribution of $\mathcal{X}_t$ **(c)** Execution time over different algorithms

explores the searching space to discover more feasible solutions for evaluation. In addition, our FF directs the GA algorithm to right track by minimizing both bandwidth usage and hop count factors. Thanks to parallel running, GA algorithm tends to examine various feasible solutions within a small time consumption.

VI. Conclusion

Network virtualization is an essential integral element of the future architecture networks (e.g. virtualized 5G networks, virtualized IoT networks), so the efficient and practical algorithms for VNE are specially demanded. In this paper, with both scalability and optimality taken into account, we proposed an intelligent parallel algorithm based on GA for online VN link embedding. Our proposed algorithm ultimately outperformed the existing approaches in all performance matrices. Brilliantly, IDPA is absolute faster than SP 44.87%, which is known as the fastest and most popular algorithm for VLiM. It is also faster than the best performance of our rivals $1,999\%$. We furthermore evaluate the execution time of parallel working model, and the statistics evidence that time complexity of our parallel algorithm is reduced to logarithmic $O(log(p))$. In future work, we will investigate the benefits of migrations between the populations of the slave nodes as well as the feasibility of GA algorithm which supports path-splitting.

References

[1] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. K. Soong, and J. C. Zhang. What will 5g be? *IEEE Journal on Selected Areas in Communications*, 32(6):1065–1082, June 2014.

[2] I. Ishaq, J. Hoebeke, I. Moerman, and P. Demeester. Internet of things virtual networks: Bringing network virtualization to resource-constrained devices. In *2012 IEEE International Conference on Green Computing and Communications*, pages 293–300, Nov 2012.

[3] Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. Rethinking virtual network embedding: Substrate support for path splitting and migration. *SIGCOMM Comput. Commun. Rev.*, 38(2):17–29, March 2008.

[4] M. Chowdhury, M. R. Rahman, and R. Boutaba. Vineyard: Virtual network embedding algorithms with coordinated node and link mapping. *IEEE/ACM Transactions on Networking*, 20(1):206–219, Feb 2012.

[5] C. Huang and J. Zhu. Modeling service applications for optimal parallel embedding. *IEEE Transactions on Cloud Computing*, 6(4):1067–1079, Oct 2018.

[6] X. Mi, X. Chang, J. Liu, L. Sun, and B. Xing. Embedding virtual infrastructure based on genetic algorithm. In *2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 239–244, Dec 2012.

[7] X.L. Chang, X.M. Mi, and J.K. Muppala. Performance evaluation of artificial intelligence algorithms for virtual network embedding. *Engineering Applications of Artificial Intelligence*, 26(10):2540 – 2550, 2013.

[8] Isha Pathak and Deo Prakash Vidyarthi. A model for virtual network embedding across multiple infrastructure providers using genetic algorithm. *Science China Information Sciences*, 60(4):040308, Mar 2017.

[9] Peiying Zhang, Haipeng Yao, Maozhen Li, and Yunjie Liu. Virtual network embedding based on modified genetic algorithm. *Peer-to-Peer Networking and Applications*, 12(2):481–492, Mar 2019.

[10] G. S. Paschos, M. A. Abdullah, and S. Vassilaras. Network slicing with splittable flows is hard. In *2018 IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pages 1788–1793, Sep. 2018.

[11] N.M. Mosharaf Kabir Chowdhury and Raouf Boutaba. A survey of network virtualization. *Computer Networks*, 54(5):862 – 876, 2010.

[12] Y. Zhu and M. Ammar. Algorithms for assigning substrate network resources to virtual network components. In *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pages 1–12, April 2006.

[13] Heinz Mhlenbein. Parallel genetic algorithms in combinatorial optimization. In OSMAN BALCI, RAMESH SHARDA, and STAVROS A. ZENIOS, editors, *Computer Science and Operations Research*, pages 441 – 453. Pergamon, Amsterdam, 1992.

[14] Frank Hansen and Gert K. Pedersen. Jensen's Operator Inequality. *Bulletin of the London Mathematical Society*, 35(4):553–564, 07 2003.