2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)
October 11-14, 2020. Toronto, Canada

A Novel One-stage Distributed Parallel Embedding for Virtualized Network Environment

Qiao Lu, Khoa TD Nguyen, ChangCheng Huang
Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Canada
Email: {qiaolu, khoatnguyen, huang}@sce.carleton.ca

***Abstract*—Network virtualization recognized as an enabling technology for the forthcoming networks is utterly popular. One of the main challenges of network virtualization is called the virtual network embedding problem. Virtual network embedding (VNE) aims to allocate a set of virtual machines onto a set of interconnected physical hardware in the cloud computing environment. Traditional exact solutions, considered as a time-consuming process to achieve a global optimal solution, have been proofed to be $\mathcal{NP}$-hard. On the other hand, some existing heuristic solutions tend to decouple VNE problems into two stages: virtual node mapping (VNoM) and virtual link mapping (VLiM). Undoubtedly, these kinds of decomposition would result in low acceptance ratio and inefficient substrate resource utilization. In this paper, we propose a distributed parallel Genetic Algorithm combined with graph theory for solving VNE in one-stage. Our proposed algorithm achieves better performance than previous baseline solutions while meeting the stringent time requirements for online VNE problems.**

I. Introduction

With the rapid development of network and cloud applications, many Internet users need to deploy new protocols or install new services on the existing network architecture. If the service provider fails to offer these communication requirements, the performance may suffer degradation dramatically. Network virtualization (NV) provisions different services among multiple virtual network requests (VNRs) through the sliceable management of the existing substrate network resources. Virtual network embedding (VNE) is one of the main tasks in the NV, which is the primary centralization of resource allocation problem. VNE aims to bring efficient resource utilization to the substrate network, and eventually a prevention of an unnecessary infrastructure expansion.

To tackle VNE problems, proposed algorithms are supposed to efficiently allocate the substrate resources for VNRs, which becomes an optimization problem with multiple rigid objectives. Solving the VNE problem is $\mathcal{NP}$-hard, hence, truly optimal solutions can be gained for small problem instances. Additionally, in most real scenarios, the VNE problem is an online problem, which requires a speedy and efficient solution. The real global optimization under dynamic demands is hard to achieve. Thus, currently, the main focus of VNE topics is on heuristic or metaheuristic approaches.

A common approach of VNE problems is to decompose the problem into two stages: mapping virtual network functions (VNFs) as virtual nodes into substrate nodes (VNoM) and mapping virtual links into one/more connected substrate links (VLiM). Although the two-step decomposition is still $\mathcal{NP}$ hard, this method reduces the computing complexity of the VNE problem in some degrees. However, separating node and link mappings may lead to neighbouring virtual nodes widely separated in the substrate network. This fact increases the cost of single/multiple paths in VLiM stage leading to low acceptance ratio and correspondingly undesirable performance. Even some approaches claim they consider the coordination between VNoM and VLiM phases, the decomposition still makes the results far from the optimal.

As mentioned in [1], with the decrease of computing cost, cloud computing becomes prevalent and can be used to conduct paralleled algorithms with high efficiency. In traditional exact methods, the $\mathcal{NP}$-hard programming is difficult to be decoupled into independent parallel tasks. Therefore, a distributed parallel meta-heuristic algorithm becomes very promising. These facts motivate us to design a fast, dynamic and one-stage meta-heuristic solution with high scalability.

Recently, research work [2] indicates that genetic algorithm (GA) achieved better performance than ant colony meta-heuristic. This achievement inspires us to investigate GA. Unfortunately, previous GA of VNE focused on either VLoM or VLiM. There is no GA-based approach proposed for VNE in one-stage. In this paper, we propose a parallel distributed GA approach which solves VNE problems in one stage with high performance and low complexity. The contributions of our proposed algorithm are summarized as follows.

- To our best knowledge, our work is the first meta-heuristic approach to solve VNE problems in one stage. As mentioned above, an one-stage mapping method keeps the integrity of the problem and avoid local optimal solution efficiently. Inspired by graph theory, the one-stage mapping could be realized in GA crossover operation in this paper.
- We utilize the natural peculiarity of easy-decomposition in the Genetic metaheuristic method, which can be run in parallel. Some variants in GA are deployed to make it tailor-made for VNE problems.
- In this paper, we proposed an unsplittable mapping method. Unsplittable and splittable mapping are equally important in network virtualization research [1], [3]. However, most of current research only concentrates on splittable mapping. Therefore, we believe our unsplittable approach is valuable for allocation techniques in recent remarkable research topics like Software Defined Network (SDN) and Edge Clouds.
- As mentioned, there is some recent research on VNE problems using GA for VLoM or VLiM phase. However, solving VNE problems in one stage is not simply combining

978-1-7281-8526-2/20/$31.00 ©2020 IEEE

VLoM and VLiM. The one-stage algorithm is far more complicated. Therefore, our one-stage GA-based solution is totally different from other previous GA algorithms.

The rest of this paper is organized as follows. Section II discusses the related work. Section III describes the network model and problem formulations. Section IV introduces our proposed approach. Section V shows our simulation results. Finally, Section VI concludes the paper.

II. Related work

Previous research proposes several heuristic algorithms in VNE problems to improve placement efficiency and global optimality. Most existing heuristic approaches apply two-stage approaches. For each stage, they tend to find one partial solution that is likely to be good.

Alternatively, there are more attention attracted by meta-heuristics which are promising for VNE problems with acceptable execution time. In addition, meta-heuristics are more likely to avoid local optimums compared with heuristic approaches. Unfortunately, most metaheuristic algorithms only focused on node mapping stage, and left link mapping stage for $k-$shortest path (unsplittable-support) or multi-commodity flow (MCF) algorithms (splittable-support). This two-stage mapping would obviously restrict the solution spaces for the link mapping stage. To our best knowledge, most of GA approaches solved VNE problems in two separate stages, specializing for node mapping [4] [5].Even in our previous study [1], a GA is proposed for VLiM stage. Nevertheless, it leaves VLoM separately solved by a greedy algorithm.

Traditional one stage VNE solutions can be achieved through Linear Programming algorithms. However, due to the high computing complexity, this kind of exact solutions are designed to solve small instances of the problem. Especially for an online dynamic VNE problems, even the exact solution cannot guarantee an optimal solution [1]. Recently, some one-stage heuristic methods are proposed. However, they still face a major problem of high time complexity to generate a greedy solution by ranking approach and breadth first search [6]. Therefore, it is of great essence to propose one-stage meta-heuristic algorithms in order to fully coordinate the virtual node and link embeddings for VNE problems.

III. Network Model

The network model is the same defined as the compared algorithms [1], [7], [8].

A. Substrate Network

We model the substrate network as a connected bidirectional graph, $G^s = (N^s, E^s)$, which consists of a set of substrate nodes N^s and a set of substrate links E^s. Each substrate node $n^s \in N^s$ is associated with capacity value $C(n^s)$ and its geographic location $loc(n^s)$. Each substrate link $e^s \in E^s$ has bandwidth capacity weight value $B(e^s)$.

B. VN requests (VNRs) and VN embedding (VNE)

The VNR is modeled as a weighted graph as $G^v(t_a, t_d, D) = (N^v, E^v, t_a, t_d, D)$. A VNR is composed of a set of virtual nodes N^v, and a set of virtual linksE^v. t_a represents arrival time of a VNR. t_d is the duration of a VNR. Each virtual node $n^v \in N^v$ in a VNR has CPU capacity requirement $C(n^v)$ and a location $loc(n^v)$. The distance between mapped substrate node and the virtual node is denoted by $dis(loc(n^v), loc(n^s))$ which should be less than D. Each link $e^v \in E^v$ has bandwidth requirement value $B(e^v)$. We also denote $\mathcal{N}^{vl}$ as the number of virtual links. Fig. 1 shows how to map a VNR into the substrate network.

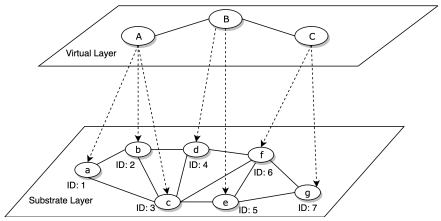


Fig. 1: An example of VNE with its associated substrate network.

To allocate the VNR, there are two kinds of constraints that we have to consider about: node capacity constraints and link bandwidth constraints. There are two constraints in node mapping:

$$C(n^v) \leqslant R^n(M(n^v)) \tag{1}$$

$$dis(loc(n^v), loc(M(n^v))) \leqslant D \tag{2}$$

where,

$$M(n^v) \in N^s, \;\; R^n(M(n^v)) \leqslant C(M(n^v))$$

$M(n^v)$ is the substrate node mapping from a virtual node n^v. $R^n(M(n^v))$ is the remaining CPU capacity of the substrate node, which is updated after allocating/releasing each VNR. We define all the substrate nodes within the distance requirement D of a virtual node n^v as the candidate nodes of n^v. We define a link mapping $M(n^v, m^v)$ from a virtual link to a substrate path. $R^e(e^s)$ is the remaining bandwidth of a substrate link e^s. $\mathcal{P}(M(n^v), M(m^v))$ is defined as a set of all substrate paths from source node $M(n^v)$ to destination node $M(m^v)$. The link mapping should follow the constraints:

$$B(e^v) \leqslant R^e(n^v, m^v) \tag{3}$$

where,

$$e^v = (n^v, m^v), \;\; M(e^v) \in \mathcal{P}^s(M(n^v), M(m^v))$$
$$R^e(e^s) \leqslant B(e^s), \;\; R^e(n^v, m^v) = \min_{e^s \in M(e^v)} R^e(e^s)$$

When a VNR mapping satisfies all the constraints above, the embedding is defined as a feasible solution.

C. Augmented Graph

Inspired by the approach adopted in [7], we also create an augmented substrate graph $G^{s'} = (N^{s'}, E^{s'})$. For each $n^v \in N^v$, we define a meta-node $\mu(n^v)$. We connect each candidate substrate node of $n^v \in N^v$ to the meta-node through a bidirectional meta-link with infinite bandwidth. We set $N^{s'} = N^s \cup \{\mu(n^v) | n^v \in N^v\}$ and $E^{s'} = E^s \cup \{(\mu(n^v), n^s) | n^v \in N^v, n^s \in N^c(n^v)\}$. $N^c(n^v)$ denotes the set of all candidate nodes for n^v. If the start and end nodes of a virtual link are n_i^v and n_j^v, $\mu(n_i^v)$, $\mu(n_j^v) \in N^{s'} \setminus N^s$ will denote the corresponding meta nodes. The augmented graph for the example in Fig. 1 and Fig. 2. With the help of the augmented graph, the node mapping and the link mapping phases could be solved coordinately in one stage.

Previous research [7] proposed to solve the one-stage mapping problem by Mix Integer Programming, which has been

observed to take too much time. This is because this technique introduces meta nodes and meta edges, which in turn increases the number of variables and constraints in the problem. In this paper, we still adopt this augmentation idea. However, we tend to deploy the augmentation in the crossover and mutation operations of the Genetic Algorithm. The novel approach not only keeps the integrity of VNE problems in one stage, but also increases the mapping speed for online dynamic placement requirement.

D. Objectives

In our proposed Genetic Algorithm (GA), we try to minimize the resource mapping costs as well as balance the load across different substrate links and nodes. The objective function in this paper is also called fitness function in GA terminology. For the load of each substrate node and link, we count all the resources (capacity or bandwidth) allocated for the virtual nodes/links. It means the fitness function takes all resource usage for the VNR as a whole, as opposed to other heuristic mapping approaches [7][8], which consider the virtual nodes/links sequentially based on a ranking method.

$$F_{NLP} = \sum_{uv \in E^S} \frac{\alpha_{uv}}{R^e(u,v) - \sum_i f^i_{uv} + \sigma} + \sum_{w \in N^S} \frac{\beta_w}{R^n(w) - \sum_{m \in N^{s'} \setminus N^s} x_{mw} C(w) + \sigma} \quad (4)$$

Existing solutions use linear objective functions [1], [7] to simplify optimization process. In our proposed algorithm, we applied the nonlinear function as shown in (4), where the bandwidth usage of the current virtual link allocations becomes part of the denominator. Previous study [9] has proved that a non-linear function has the similar computing complexity with a linear function, but it has better performance than the same algorithm using linear fitness. In equation (4), σ is a small positive constant to avoid the denominator becoming zero. f^i_{uv} describes the total amount of flow from u to v for the ith virtual link under the specific mapping scenario. And x_{mw} denotes a binary variable, which has the value 1 if the meta link is activated; Otherwise, it is set to 0. $0 \leqslant \alpha_{uv} \leqslant R^e(u,v)$ and $0 \leqslant \beta_w \leqslant R^n(w)$ are parameters to control the importance of load balancing while mapping the request. Obviously, a fitness function like (4) improves the resource efficiency by avoiding a small residual substrate resource which may cause resource fragmentation.

IV. Proposed One-stage VNE Algorithm

Nowadays, cloud computing is becoming prevalent. With the decrease of computing cost, people concern more on execution time rather than computing cost. Parallel and distributed computing has recently emerged as an effective mechanism to tackle large and complex problems with less time consuming and lower cost by supporting the concurrency. Nevertheless, how to design the parallel structure for VNE problems is still a stringent problem. Thanks to previous research [10], it has proved that parallel computing can be applied naturally to GA to reduce the execution time. Instead of achieving partial parallelism (parallel link mapping) in [1], We improve the GA-based VNE solution by realizing both node and link mapping in one-stage, thereby fully parallelizing VNE problems.

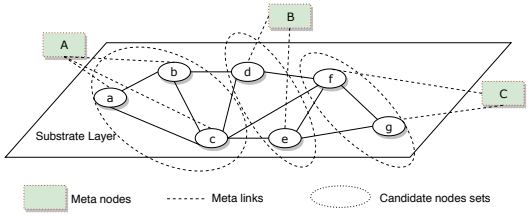


Fig. 2: The augmented graph of the example in Fig 1.

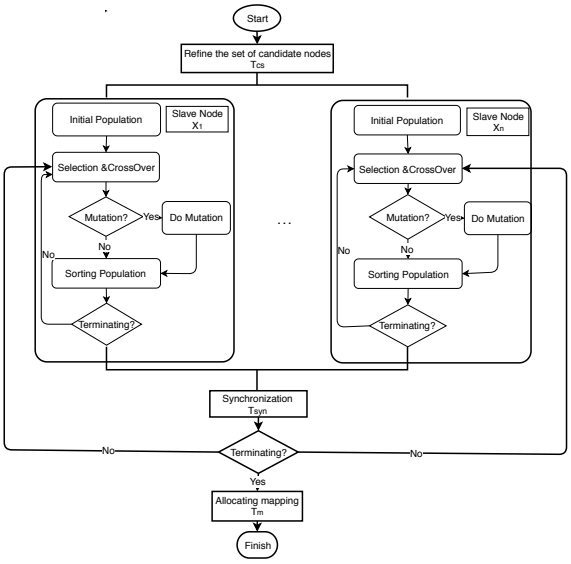


Fig. 3: Parallel execution flow chart

Our proposed parallel Genetic Algorithm can be run over many machines in a distributed way as shown in Fig. 3. At the beginning, the set of candidate nodes will be refined in the master working node and then sent to slave working nodes as the inputs. Each parallel slave working machine runs independently and generate descendant solutions. The process of GA in each slave machine starts with the selection of an initial population. The slave working machines use crossover and mutation operations to produce offspring which inherit the characteristics of their parents. The produced offspring are added and sorted into the next generation. If parents have better fitness values which are calculated through our objective function in (4), their offspring have a better chance to survive. This process keeps on iterating and at the end, the fittest individuals are achived. The fittest individual amongst all slave machines will be selected as the final solution.

A. Refine the sets of candidate nodes

As described in section III, we define the set of all candidate nodes for a virtual node as $N^c(n^v)$. The set includes all substrate nodes within the maximum distance of the virtual node. However, the constraints to map a virtual node are not only related to the distance factor, but also concerning the remaining capacity of the substrate nodes as shown in (1) and (2). In this step, we try to restraint the set of candidate nodes for each virtual node by checking the constraint in (1). Its

initiative is to avoid infeasible substrate nodes entering the population and make the genetic production more effective.

B. Parallel slave procedure

After the candidate nodes have been refined by node mapping constraints, all the slave nodes start the Genetic procedures independently.

1) Genetic representation: In our problem, a request consists of several virtual links, which connect virtual nodes together. Our proposed GA encoding denotes each gene as a virtual link mapping method based on augmented graph. That is to say, a gene is started and ended by meta links. Each chromosome acts as a VNR mapping solution.

Specifically, the process begins with a set of chromosomes/solutions which is called a population. A chromosome $\boldsymbol{c_i}$ denoted by (5) represents a feasible VNR solution. A gene $\boldsymbol{g_{ij}}$ is a mapping solution for the corresponding virtual link. Similarly, A node denoted by n_{ijk} represents a substrate/virtual node. The first two subscripts indicate its gene, and the third one denotes its position in the gene. A gene can be denoted by (6) with a variable length d_{ij}. The first and the last links in a gene represent the node mapping results while the intermediate links indicate the link mapping solutions. Therefore, the first and the last nodes of a gene should be meta nodes. Each gene $\boldsymbol{g_{ij}}$ can be divided into two partial paths as (7): head $\boldsymbol{H_{ijk}}$ and tail $\boldsymbol{T_{ijk}}$.

$$\boldsymbol{c_i} = \{\boldsymbol{g_{i1}}, \boldsymbol{g_{i2}}, ..., \boldsymbol{g_{ij}}, ..., \boldsymbol{g_{i\mathcal{N}^{vl}}}\} \tag{5}$$

$$\boldsymbol{g_{ij}} = \{n_{ij1}, ..., n_{ijk}, ..., n_{ijd_{ij}}\} \tag{6}$$

$$\boldsymbol{g_{ij}} = \left[\boldsymbol{H_{ijk}}, \boldsymbol{T_{ijk}}\right], \ \ \forall k \in (0, d_{ij}) \tag{7}$$

where,

$$\boldsymbol{H_{ijk}} = \left[n_{ij1}, n_{ij2}, \ldots, n_{ijk}\right]$$

$$\boldsymbol{T_{ijk}} = \left[n_{ij(k+1)}, n_{ij(k+2)}, \ldots, n_{ijd_{ij}}\right]$$

2) Initial Population: The purpose of the initialization is to generate a population of M chromosomes, where each chromosome is a feasible embedding solution. To select a chromosome, we need to choose genes that form a feasible solution. This is done in two steps. The first step is to randomly choose the substrate node for each virtual node from the set of candidate nodes. After all the substrate nodes have been selected, all the meta links are set in a chromosome. The second step is to look for a substrate path of each gene. To find some good potential substrate paths, we introduce the shortest path pool [1] which can be done before any online VNR arrives. For each source-destination pair in the substrate network, we identify K shortest paths as our path pool.

$$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{c_1} \\ \boldsymbol{c_2} \\ \vdots \\ \boldsymbol{c_i} \\ \vdots \\ \boldsymbol{c_M} \end{bmatrix} = \begin{bmatrix} \boldsymbol{g_{11}} & \cdots & \boldsymbol{g_{1j}} & \cdots & \boldsymbol{g_{1\mathcal{N}^{vl}}} \\ \boldsymbol{g_{21}} & \cdots & \boldsymbol{g_{2j}} & \cdots & \boldsymbol{g_{2\mathcal{N}^{vl}}} \\ \vdots & \ddots & & \ddots & \vdots \\ \boldsymbol{g_{i1}} & \cdots & \boldsymbol{g_{ij}} & \cdots & \boldsymbol{g_{i\mathcal{N}^{vl}}} \\ \vdots & \ddots & & \ddots & \vdots \\ \boldsymbol{g_{M1}} & \cdots & \boldsymbol{g_{Mj}} & \cdots & \boldsymbol{g_{M\mathcal{N}^{vl}}} \end{bmatrix} \tag{8}$$

After all candidate genes in a chromosome have been selected, we should check the feasibility of the chromosome as described in Section III-B. Only the feasible chromosome is added to the population denoted by $\boldsymbol{P}$. This process continues until a feasible chromosome is selected. $\boldsymbol{P}$ can be described as a matrix shown in (8).

3) Selection and Crossover: We select one-pair parents from the population as the inputs of crossover operation. Each gene in a chromosome conducts the crossover with its corresponding gene in the other parent chromosome. We denote two parent chromosomes as $\boldsymbol{c_s}$, $\boldsymbol{c_r}$ and two generated children chromosomes as $\boldsymbol{c_{(M+1)}}$, $\boldsymbol{c_{(M+2)}}$. Each gene in the chromosome should crossover with the counterpart of the other parent chromosome. If there is a node n_{sju} in $\boldsymbol{g_{sj}}$ is equivalent to a node n_{rjv} in $\boldsymbol{g_{rj}}$, where u and v are not the indexes of source or destination node, we denote the node as a common node. There are two scenarios in the crossover operation:

a) : There are one/more common nodes in parent genes. We select one common node and utilize it as an intermediate node. The partial parts of the two genes are swapped as shown in (9) and (10).

$$\boldsymbol{g_{(M+1)j}} = \left[\boldsymbol{H_{sju}}, \boldsymbol{T_{rjv}}\right] \tag{9}$$

$$\boldsymbol{g_{(M+2)j}} = \left[\boldsymbol{H_{rjv}}, \boldsymbol{T_{sju}}\right] \tag{10}$$

b) : There is no common node in two-parent genes. In this scenario, a crossover link is selected in each gene. To make sure the children's genes are still valid paths, the partial paths are obtained from the shortest path pools and connect the genes. For example, we randomly identify a crossover link $(n_{sju}, n_{sj(u+1)})$ for $\boldsymbol{g_{sj}}$, and select $(n_{rjv}, n_{rj(v+1)})$ for $\boldsymbol{g_{rj}}$. A substrate path $P_{n_{sju}, n_{rj(v+1)}}$ between n_{sju} and $n_{rj(v+1)}$ is selected from the source-destination path pool. The results of crossover without common node are shown in (11) and (12).

$$\boldsymbol{g_{(M+1)j}} = \left[\boldsymbol{H_{sju}}, P_{n_{sju}, n_{rj(v+1)}}, \boldsymbol{T_{rj(v+1)}}\right] \tag{11}$$

$$\boldsymbol{g_{(M+2)j}} = \left[\boldsymbol{H_{rjv}}, P_{n_{rjv}, n_{sj(u+1)}}, \boldsymbol{T_{sj(u+1)}}\right] \tag{12}$$

After crossover, the child gene may contain loops, which is an invalid path. Therefore, each gene has the validation check to remove loops. Then it is the time to compose a chromosome. In the proposed algorithm, there is only one way to compose the children chromosomes based on the node mapping results. Specifically, the parent genes swap partial nodes and generate the children genes. This operation intermingles the node mapping combinations if the meta links are exchanged between parent genes. Thus, we have to carefully organize the children genes and make sure a virtual node mapped into the same substrate node in different genes of a child chromosome.

In graph theory, the composition of a chromosome becomes a matching problem [11]. It means that, for a given VNR, a matching $\mathcal{M}$ in the VNR graph is a set of pairwise non-adjacent edges, none of which are loops. The matching problem could be explained as the graph coloring problem, which requires different colors for the all adjacent nodes. A special case in the matching problem is that the request is a bipartite graph, whose nodes can be divided into two disjoint and independent sets. Determining whether or not the graph is bipartite is computable in linear time using breadth-first search or depth-first search. Taking Fig. 1 as an example, we select the meta links in parent chromosomes, the children genes could be integrated into children chromosomes without node mapping disorder since it is a bipartite VNR graph.

However, the VNR graph sometimes, is not a bipartite graph. For example, a request contains a triangle: there is a extra virtual link between A and C for the request in Fig. 1. To

solve this issue, a set of meta links is built to record the node mapping results. Once a meta link is set for one virtual link, the following crossover must follow the restriction to map the virtual node based on the recorded meta link .

4) Mutation: The mutation operation is essentially aimed for expanding the solution space and broadening the search. Each gene in the children chromosomes go through mutation operation with a fixed probability called mutation rate. In our proposed algorithm, both of the node mapping and link mapping are encoded to the genes. If the node mapping is mutated, all genes in a child chromosome have to be updated. Therefore, we have two steps in the mutation operation.

The first step is node mapping mutation that checks if a meta link is included between the mutated nodes. A meta link represents the mapping of a virtual node to a substrate node. We have to update the node mapping over the whole chromosome if meta link mutation happens. To mutate a node mapping of n^v, an alternative node n_a^s from our refined set of candidate nodes $N^c(n^v)$ is selected to replace the current mapping node n_c^s. We evaluate the new node n_a^s and the non-meta mutated node as the updated mutated nodes.

Before the first step finishes, we have to find all other genes including $\mu(n^v)$ and update all corresponding meta links. As we know the genes are valid paths in the augmented graph, hence, if we change one node in the gene, we have to make sure the gene is still a connected path. Therefore, we mark the new node n_a^s and its neighbouring substrate node as the updated mutated nodes for the next step. Then the mutation operation goes to the second step: path mutation. An alternative path chosen from the path pool replaces current partial route and connects two updated mutated nodes.

5) Sorting and terminating condition: The children chromosomes generated and mutated by previous operations are added into the population $\boldsymbol{P}$. Only the best M chromosomes are kept in the population, then the slave procedure goes back to select parent chromosomes from the population for the next generation. This reproduction stops either the best solution has not been changed in t_{suc} successive times or the number of generations has reached a fixed number t_{max}.

C. Synchronization and Allocation

After all slave procedures have obtained the solution, all results are synchronized and the best placement mapping is found. The last step is to allocate the VNR into the substrate network according to the mapping solution.

D. Execution time analysis

With the computing cost deceasing and online network resources demands increasing, we propose a parallel distributed GA framework to achieve fast, efficient online embedding. In this framework, the parallel level (the number of slave procedures) can be tuned according to the trade-off between available computing resource and expected performance. In this section, we concentrate on how much time could be saved in the proposed architecture. We first consider all slave procedures running sequentially. The total time consuming for sequential running can be calculated by the sum of each parallel slave procedure. Therefore, the complexity for sequential running grows linearly along with the parallel level.

For parallel running scheme as shown in Fig. 3, except the time consuming on refining the set of candidate nodes (T_{cs}) and the time on synchronization (T_{syn}), all the procedures can be executed in parallel. The execution time denotes the maximum execution time among all parallel nodes for the reason that the parallel algorithm should wait until the slowest parallel procedure finishes its task. Hence, the total parallel execution time is the sum of the slowest parallel procedure's execution time and the master procedures' execution time. The master procedures' execution time, through the simulation, accounts for 0.9% of the total parallel execution time on average. We approximate the total parallel execution time as the execution time of the slowest parallel procedure. We evaluate the upper bound of mean value of the total parallel execution time according to Cramer-Chernoff method and Jensen's inequality. Finally, we found the parallel running scheme can improve the execution time from linear time to logarithmic time. The detailed derivation can be found in [1].

V. Performance Evaluation

In this section, we present the performance of our proposed algorithm. To make the comparison fairly, we set the simulation with the same evaluation environment of other previous compared algorithms.

A. Simulation settings

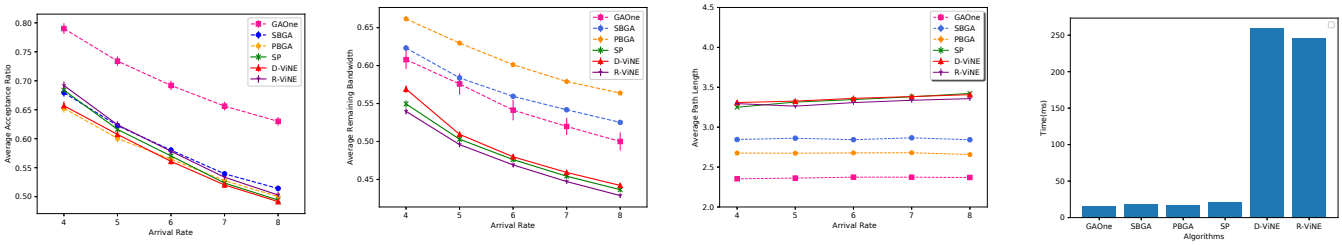
To compare the algorithms fairly, we use the same simulation setting in previous research [1] [7]. We randomly generate three different substrate network typologies with 50 nodes in 25×25 grids by the waxman algorithm. The capacity and bandwidth resources of the substrate networks are real numbers uniformly distributed. We assume that the VNRs dynamically arrive in a Poisson distribution.

B. Comparison Method

In this paper, we compared our proposed one-stage approach GAOne with five algorithms. D-ViNE and R-ViNE [7] are considered as the benchmark algorithms of mapping virtual networks. Another algorithm in the comparison list is G-SP [8], which deploys a greedy node mapping and the shortest path algorithm for link mapping. We choose it for the comparison because the shortest path method is widely used in several meta-heuristic algorithms, and also it is considered the fastest algorithm due to its simplicity. We also compared PBGA and SBGA algorithms [1], which adopt a greedy approach for VLoM stage and a path based Genetic Algorithm (PBGA) and a segment based Genetic Algorithm (SBGA) for VLiM stage. Comparing with [1], we can clearly learn about how much our one-stage algorithm improves the performance.

C. Evaluation Results

The performance metrics that we measure in this simulation are the average acceptance ratio, the average remaining bandwidth percentage and the total execution time. We pay more attention on link utilization analysis than node utilization, since we assume the node mapping is unsplittable. The acceptance ratio in Fig. 4a indicates the node utilization. The link mapping solution generally contains multiple substrate paths, thereby deserving more attentions in this paper. Fig.4 describes the



(a) Average acceptance ratio over arrival rates

(b) Average remaining bandwidth over arrival rates

(c) Average path length over different algorithms over arrival rates

(d) Total execution time analysis

Fig. 4: Performance comparison

average value over arrival rates from 4 to 8 per 100 time units with 95% CI (confidence interval).

We measure the execution time for each procedure as shown in the Fig. 3. We set the parallel level as 16. According to the convergence analysis [1], the acceptance ratio reaches converged values when the parallel level goes up to 16. Definitely, the parallel level could be adjusted freely in terms of a time limitation or a high acceptance requirement.

We summarize the observation of results as following:

1) As shown in Fig.4a, GAOne has the best acceptance ratios. This observation implies that our GA with a comprehensive objective function can generate more efficient solutions. Moreover, the results verify the benefits of one-stage strategy.
2) The greedy solutions (G-SP, D-ViNE, R-ViNE) occupy more resources in Fig.4b, while the load balanced solutions (GAone, SBGA, PBGA) utilize the bandwidth more efficiently. Moreover, the utilization of the bandwidth is increased in GAone. The higher acceptance ratio of GAone than SBGA and PBGA comes from the higher bandwidth utilization.
3) The average path length in Fig. **??** depicts the bandwidth saving for load balanced algorithms. They take less average path lengths, and save more bandwidth resource for future VNRs. Another observation is that GAOne consumes more bandwidth than other load balanced algorithms, but it has the shortest average path length. Since it accepts more requests as illustrated in Fig. 4a, it consumes more bandwidth resource.
4) As shown in Fig. 4c, D-ViNE and R-ViNE take more time to generate the solution due to their relaxed linear programming mechanism. The G-SP performs the greedy VLoM and greedy shortest path VLiM, which simplifies the problem with small consuming time. However, G-SP still takes longer time than the distributed parallel solutions.

Among the parallel algorithms, SBGA and PBGA essentially realize a partial parallelism as they still run VLoM stage sequentially. The average execution time is 18.74ms for SBGA and 16.62ms for PBGA. SBGA consumes longer time because it considers the path restructuring while PBGA is a path based algorithm. Our proposed algorithm GAOne considers the node and link restructing in one-stage and takes 16.05ms on average. The consuming time is smaller than both SBGA and PBGA by reason of one-stage full parallel running scheme.

VI. Conclusion

There is few existing research discussing one-stage online VNE problems due to the high complexity. In this paper, we propose a one-stage distributed parallel Genetic Algorithm which takes both scalability and optimality into account. We realize the one-stage mapping for both virtual nodes and their associated links through the genetic reproduction mechanism in combination with graph theory. Our intensive simulation shows that GAOne algorithm achieves a better performance both on acceptance ratio and time saving aspects. Besides, the results confirm our argument that the one-stage mapping greatly broadens the solution space and provides more efficient mapping results. We also extend our previous research [1] by realizing full parallelism of Genetic Algorithm. The new distributed parallel structure saves more time for online VNE problems, which is significantly important for the 5G network, IoT demands and future SDN deployment.

References

[1] Qiao Lu, Khoa Nguyen, and Changcheng Huang. Distributed parallel algorithms for online virtual network embedding applications. *International Journal of Communication Systems*, page e4325, 2020.
[2] 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.
[3] G. S. Paschos, M. A. Abdullah, and S. Vassilaras. Network slicing with splittable flows is hard. In *2018 IEEE 29th Annual International Symposium on PIMRC*, pages 1788–1793. IEEE, 2018.
[4] 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, 2019.
[5] 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, 2017.
[6] H. Cao, S. Wu, Y. Guo, H. Zhu, and L. Yang. Mapping strategy for virtual networks in one stage. *IET Communications*, 13(14), 2019.
[7] M. Chowdhury, M.R. Rahman, and R. Boutaba. Vineyard: Virtual network embedding algorithms with coordinated node and link mapping. *IEEE/ACM Transactions on Networking (TON)*, 20(1):206–219, 2012.
[8] M. Yu, Y. Yi, J. Rexford, and M. Chiang. Rethinking virtual network embedding: substrate support for path splitting and migration. *ACM SIGCOMM Computer Communication Review*, 38(2):17–29, 2008.
[9] Q. Lu and C. Huang. Distributed parallel vn embedding based on genetic algorithm. In *2019 IEEE Symposium on Computers and Communications (ISCC)*, 2019.
[10] H. Mühlenbein. Parallel genetic algorithms in combinatorial optimization. In *Computer science and operations research*. Elsevier, 1992.
[11] L. Lovász and M. Plummer. *Matching theory*. American Math. Soc.