GAONE: A Novel Approach for Online One-stage Virtual Functions Placement

Qiao Lu[1], Khoa TD Nguyen[1], and Changcheng Huang[1]
[1]Carleton University, Ottawa, ON, Canada

Recently, Network Function Virtualization (NFV) has gained a lot of attention as an enabling technology of the future inter-networking paradigm. By decoupling substrate network equipment from network functions that run on them, NFV helps Telecommunication Service Providers reduce Operating Expenses (OPEX) and Capital Expenses (CAPEX). One of the main challenges of NFV deployment is the optimal Virtual Network Function (VNF) placement in the network infrastructure in a suitable way. This is called a VNF-Forwarding Graph Embedding (VNF-FGE) problem, which is considered as a generalization of the virtual network embedding (VNE) problem. [1]. Previous research formulates VNF placement as a mixed integer linear programming problem that has been proved to be $\mathcal{NP}$-hard. On the other hand, some heuristic approaches are proposed to simplify the VNF placement into two separate stages. However, this separation overlooks the coordination between virtual functions and associated virtual links, thereby resulting in low acceptance ratio and inefficient substrate resource utilization. To address the limitations of previous approaches, we propose a distributed parallel Genetic Algorithm that is combined with graph theory for solving VNF placement in one-stage. Extensive simulations have shown that our proposed algorithm achieves better performances compared to previous baseline solutions, while meeting the stringent time requirements for online VNF placement applications.

***Index Terms*—Network Function Virtualization, VNF placement, Virtual Network Embedding, Genetic Algorithm, Graph Theory, Parallel Distributed Algorithm.**

I. Introduction

TRADITIONAL networks are rooted in fixed-function physical infrastructure (e.g. switches or routers). There are certain functions largely implemented in these dedicated network devices to support the networking, which makes the traditional networks lack flexibility to adapt network requirement changes. Therefore, network virtualization (NV) has been proposed to overcome the resistance of the traditional networks to the fundamental change. Virtualization of computational and storage resources is widely used in cloud computing, however, virtualization in network functions has proven to be a lot more challenging [2].

To address this challenge, Network Function Virtualization (NFV) has emerged as a new approach to design, deploy and manage network services. The idea of NFV is decoupling the network functions from physical network equipment. This 'virtualization' makes service deployment more flexible and scalable [3]. In the NFV architecture, the traditional Internet Service Providers are decoupled into Services Providers (SPs) and Infrastructure Providers (InPs). The SPs are responsible for deploying and offering customized network services to end users, while InPs are in charge of maintaining and allocating physical resources to different SPs. A SP generally operates and maintains virtual networks (VNs) leased from substrate networks (SNs) of the InP according to a virtual network embedding (VNE) strategy. This means a VN is made up of a set of virtual network functions (VNFs), which could be initialized and terminated on physical devices in runtime. An example of allocating a VN into the SN is described in Fig. 1.

To achieve fast, scalable and dynamic composition and allocation of VNFs, it becomes the NFV resource allocation (NFV-RA) problem. NFV-RA is carried out in three stages [1]:

Corresponding author: Qiao Lu (email: qiaolu@sce.carleton.ca).

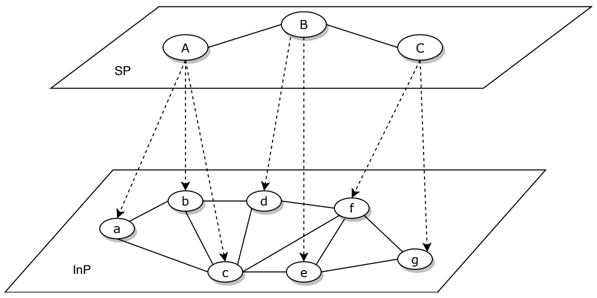

Fig. 1: An example of a VN with its associated SN.

1) VNFs Chain composition (VNFs-CC), 2) VNF Forwarding Graph Embedding (VNF-FGE) and 3) VNFs Scheduling (VNFs-SCH). Specifically, VNF-FGE, acting as an important step in NFV-RA, seeks where to allocate the VNFs in the network infrastructure in a suitable way. VNF-FGE [1] can be considered as a generalization of the well-known VNE, which aims to enable high volume, large scale, and agile network services. The VNE problem can be considered as a multi-objective optimization problem (e.g. lower network cost, long-term profits) with several stringent constraints, and solving the VNE problem is $\mathcal{NP}$-hard even in an offline setting [4]. An offline setting handles a set of VN requests (VNRs) that are indeed known in advance in a static network topology. However, in most real-life scenarios, VNE is an online problem, which requires a speedy and efficient solution. Additionally, a global optimality under dynamic demands is hard to achieve since its solution normally do not try to reallocate the already-mapped requests in the past to avoid possible network disruptions to the existing services. Moreover, the unknown future requests are difficult to anticipate. Most of the current NFV-RA approaches are based on heuristics that attempt to place online VNFs in

polynomial time by sacrificing some degrees of optimality.

To simplify the allocation of virtual resources, the contemporary research proposals mostly decouple the VNE problem into two separate stages: mapping VNFs as virtual nodes into substrate nodes – virtual node mapping (VNoM) and mapping virtual links into substrate paths with multiple connected substrate links – virtual link mapping (VLiM).

The two-step decomposition approaches can simplify the algorithmic complexity. However, the complexity is still $\mathcal{NP}$ hard. Moreover, these two-stage algorithms might result in the situation that virtual neighbouring nodes are mapped onto the substrate nodes that are probably far from each other. The long distance between substrate nodes may lead to unexpected longer substrate paths and network defragmentations. This fact is consequently caused by an inefficient VNE solution that declines the substrate resource utilisation and the acceptance ratio in the long term. Although some approaches [5] are claimed to consider the coordination between VNoM and VLiM stages, the achieved results are still far from the optimal.

A VNE approach can be classified into either splittable or unsplittable. VNE under splittable and unsplittable configurations is a principle research topic in Software Defined Network (SDN), NFV and Future Edge Clouds. A splittable mapping allows to split a virtual node/link over multiple substrate nodes/paths in the SN. Indeed, splitting mapping enables better resource utilization in theoretical analysis by gathering small pieces of available substrate resources, thereby increasing the acceptance ratio. However, such method has many implementation issues in practice [6]. On the other hand, there are limited research on the unsplittable mapping. In most previous work, unsplittable mapping algorithms, formulated as Integer Programming problems, are still $\mathcal{NP}$-hard. The major challenge of unsplittable embedding is again its complexity issue. Several VNE algorithms apply a heuristic approach for VNoM and the k-shortest path for VLiM stage. Although such decomposition can compromise the time complexity, these heuristic solutions cannot guarantee optimal or even near-optimal mapping results. In this paper, we propose a one-stage solution to address previous unsplittable embedding limitation on time complexity.

Recently, some online VNE approaches claim that their approaches [7], [8] could complete in polynomial time. However, the approaches still take longer time than the industry expectations on an online embedding system. Current VNE approaches have to make a compromise between the mapping performance (e.g., acceptance ratio and network utilization) and the embedding execution time. Based on the above concerns, an unsplittable, one-stage, heuristic and efficient online VNE solution is in sore need of further study. In this paper, by introducing the augmented graph and combining graph theory, we propose a tailored Genetic Algorithm (GA) that coordinately solves the VNE problem in one-stage as well as reducing the embedding time complexity by deploying a distributed parallel architecture. To the best of our knowledge, this is the first paper that applies a novel metaheuristic GA, namely GAOne, for tackling the VNE problem in one-stage.

We conclude our contributions as follows:

First of all, most of the VNE heuristic algorithms implement the k-shortest path mechanism for link mapping, no matter what the objectives are. The k-shortest path is a greedy approach that likely produces a good solution but it cannot efficiently handle the rigid multiple objectives such as accumulated long-term revenue, low network cost, improved resource utilization, high acceptance ratio and so on. In our proposed algorithm, the coordination between node and link mappings are object-oriented. Only the solution with highest fitness value can "survive" in the GA's procedures. Besides, during the evolutionary process, the embedding solution of virtual nodes combined with associated link mapping can be generated in the crossover and mutation operations in GA, which is able to avoid local optima.

Secondly, a number of heuristic/metaheuristic algorithms simplify VNE problems by separating steps, even for some one-stage algorithms. These algorithms solve the partial problem by separating node and link stages or iterating virtual nodes. We will discuss this part in detail in Section II. In this paper, we consider the whole embedding solution as a single feasible solution and then evolve feasible solutions through GA operations. The fitness value of each feasible solution is calculated with the coordination of all virtual nodes and links.

Last but not least, one-stage solutions usually accompany with the cost of time complexity due to their complicated mathematical models. Our proposal tackles this severe problem by a distributed and parallel scheme. The distributed and parallel computing has recently emerged as an effective mechanism to handle large and complex problems with less time consuming and lower cost. However, a VNR includes several VNFs and virtual links that are highly dependent on each other making parallelism complicated, so finding an efficient one-stage VNE algorithm is literally challenging. In our proposed GA, the genetic operations are conducted among different feasible solutions that are independent and mutually exclusive. GA generally seeks for the best solution by evaluating and improving multiple feasible solutions through its evolution processes.

In summary, our main objective is to propose a coordinated one-stage VNE strategy in the NFV environment to improve resource utilization and deploy our approach in a distributed and parallel framework to meet the time requirement in an online scenario. The rest of this article is organized as below: the related work is described in Section II. The network model and objective function are presented in Section III. In Section IV, our proposed GAOne is detailed. We illustrate our performance evaluation results and discussions in Section V and finally we conclude the paper in Section VI.

II. Related work

A. *Previous literature*

VNE problems have attracted many researchers' attention in the past decade. Since VNE is known as a $\mathcal{NP}$-hard problem, some exact methods formulated VNE problems as integer linear programming (ILP) or mixed integer linear programming (MILP), bringing an exponential run-time cost. Consequently, most of the prior studies have proposed heuristic solutions to reduce the time complexity. Existing literature experiences several heuristic VNE algorithms proposed to achieve global optimality as well as embed each given VN in polynomial time. However, the majority of these approaches address the VNE problem in two stages consisting of embedding all virtual nodes and virtual links separately. Each stage tends to get a partial solution without any performance guarantee.

Generally, two-stage unsplittable VNE algorithms involve a greedy node embedding and a subsequent k-shortest path (SP) link mapping [9], [10], or a ranking-based node embedding and again the k-shortest path (SP) link mapping [11], [12], [13]. Some metaheuristic methods are also proposed in VNE problems, for example, [14] presents a particle swarm optimization to achieve a better local selection strategy for VNoM by adjusting possible VNE solutions and then the k-shortest path (SP) is deployed for VLiM stage.

In terms of one-stage proposals, [5] formulated the VNE problem as a pure MILP model. Due to the computing complexity, the authors applies an integer relaxation so that the solution could be obtained in polynomial time. However, the relaxation variant is executed in two stages and the embedding results of the relaxed model are considered as node mapping outcome.

Di et al. in [15] proposes a one-stage VNE solution that coordinates node mapping and link mapping. At each step, a virtual node is mapped according to the already-mapped virtual nodes and the connecting virtual links among mapped virtual nodes. Therefore, the link embedding is considered during node mapping. Besides, this approach allows backtracking steps that can improve the probability of successful mapping by iterating over more possible mappings.

The authors in [8] deployed a heuristic method (CAN-A) to construct the candidate substrate node subset and the candidate substrate path subset before conducting the ILP-based mapping. Through the time complexity analysis in [8], the approach reduces the execution time compared with the pure MILP solution. However, it takes longer time than the mentioned approaches [5]. Hence, these one-stage solutions can be gained for small problem instances. With the network scales increase, the embedding time still grows exponentially, which is impractical for the online VNE problem.

Similarly, [7] deploys an one-stage heuristic mapping algorithm (VNE-RTOS). For each VNR, VNE-RTOS adopts a node ranking for both VN and SN, then conducts the greedy embeddings for the first two highest ranked-value virtual nodes and eventually maps the corresponding virtual links between these nodes using SP method. This procedure iterates untill all virtual nodes and links have been mapped. Indeed, these approaches [15], [7] perform the coordinated mapping between virtual nodes and virtual links. However, they neglect the coordinations between virtual nodes. They map one virtual node at a time in accordance with specific heuristic node ranking methods. In fact, these method still separate the mapping procedure into multiple virtual nodes' stages. Hence, these are not pure one-stage algorithms in essence.

With respect to metaheuristic algorithms proposed to tackle VNE problems, most embedding solutions are only focused on node mapping stage, and leaving link mapping stage for $k-$shortest path (unsplittable-support) or multi-commodity flow (MCF) algorithms (splittable-support). This two-stage mapping would certainly restrict the solution spaces for the link mapping stage. To our best knowledge, most of the GA-based VNE solutions are conducted in two separate stages, specializing for node mapping [16] [17] [18]. Even in our previous study [19], a GA is merely focused on the link mapping stage, that leaves the node mapping solved separately by a greedy algorithm.

B. Summary

To summarize, most heuristic algorithms solve the VNE problem in two separate stages, using relaxed ILP model or greedy methods in node mapping stage and deploying the link mapping by k-shortest path approach. The two-stage methods provide sub-optimal solutions for each stage. This separation lead to an unguaranteed result. Even some one-stage heuristic algorithms still deploy the k-shortest path approach for link mapping; however, these approaches still solve the VNE problem in separate steps. For exact VNE algorithms, they are realized in a pure one-stage operation and the majority of them usually utilize the MILP/ILP model, causing a non-polynomial execution time. Previous pure one-stage solution cannot meet the time requirement for an online scenario.

III. Network Model

THIS section elaborates on the VNE network model and VNE problem descriptions. The main index notations throughout this paper are listed in Table I.

A. Network Model Description

1) Substrate Network

In VNE problem, researchers usually model the underlying SN by using the bidirectional weighted graph $G^s = (N^s, E^s)$. The SN infrastructure is composed of a set of substrate nodes N^s (e.g. hosts, routers) and a set of substrate bidirectional links denoted as E^s (e.g. twisted pairs).

Each substrate node $n^s \in N^s$ is characterized by its node capacity $C(n^s)$, and its geographic location $loc(n^s)$. The location of a substrate node is defined on x and y coordinates in this paper. The substrate link $e^s_{mn} \in E^s$ indicating the link between substrate node m^s and n^s has a finite bandwidth $B(e^s_{mn})$. $B(e^s_{mn})$ is always equal to $B(e^s_{nm})$ due to the nature of bidirectional substrate graphs. In addition, we use $P^s(m^s, n^s)$ to represent the set of all substrate path from node m^s to n^s. p_{mn} is one path selected from the path set $P^s(m^s, n^s)$.

2) Virtual Network

Conventionally, a VN consists of a set of dedicated network service boxes such as firewall, load balancers and application delivery controllers that are concatenated together to support a specific application [20]. The VNR is modeled as a weighted graph, denoted by $G^v(t_a, t_d, D) = (N^v, E^v, t_a, t_d, D)$. N^v is a set of virtual nodes, whilst E^v denotes a set of virtual links. t_a is arrival time of the VNR. t_d is the duration of the VNR and D represents the maximum distance between a virtual node and its associated substrate node. Each virtual node $n^v \in N^v$ in a VNR has a CPU capacity requirement $C(n^v)$ and a desired virtual node location $loc(n^v)$. The distance between a substrate

TABLE I: Mathematical notations used in the paper

Variable	Description
N^s	a set of substrate nodes
E^s	a set of substrate links
m^s, m^s	a substrate node
e^s_{mn}	a substrate link
$C(n^s)/C(n^v)$	the CPU capacity of n^s/ the CPU requirement of n^v
$B(e^s_{mn})/B(e^v_{mn})$	the bandwidth capacity of e^s_{mn}/ the bandwidth requirement of e^v_{mn}
$loc(n^s)$	geographical location of n^s
$P^s(m^s, n^s)$	a set of substrate paths from m^s to n^s
p_{mn}	a substrate path from m^s to n^s
n^v, m^v	a virtual node
e^v_{mn}	a virtual link between m^v and n^v
N^v	a set of virtual nodes
E^v	a set of virtual links
D	maximum distance between a virtual node and its associated substrate node
$dis(loc(n^v), loc(n^s))$	the distance between two nodes
$\mathcal{N}^{vl}$	the number of virtual links in current request virtual network
$\mu(n^v)$	the meta node of n^v
$N^c(n^v)$	the set of all candidate substrate nodes of n^v
$\mathcal{M}(n^v)$	the substrate node mapping of n^v
$R^n()$	the current remaining node capacity function
$\mathcal{M}(e^v_{nm})$ $R^e()$	the substrate link mapping of e^v_{nm} the current remaining link capacity function
E^v_c	the set of all virtual links belonging to current allocated VNRs
r_b	remaining bandwidth ratio
r_a	VNR acceptance ratio

node and the virtual node is denoted by $dis(loc(n^v), loc(n^s))$. Each link $e^v_{mn} \in E^v$ has bandwidth requirement value $B(e^v_{mn})$ that indicates the required bandwidth between virtual node m^v and n^v. We also denote $\mathcal{N}^{vl}$ as the number of virtual links.

3) Augmented Network

Inspired by the approach adopted in [5], we also create an augmented substrate graph $G^{s'} = (N^{s'}, E^{s'})$ for the graph $G^s = (N^s, E^s)$. For each $n^v \in N^v$, a corresponding meta-node $\mu(n^v)$ is created. Each candidate substrate node of $n^v \in N^v$ is connected 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. The augmented graph for the example in Fig. 1 is shown in Fig. 2. With the augmented graph, the node mapping and link mapping phases of the VNE problems can be solved coordinatedly in one stage.

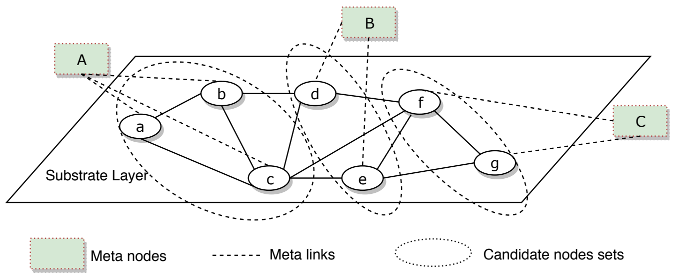

Fig. 2: The augmented graph of the example in Fig 1.

Previous research [5] deploys the augmented graph to solve the one-stage mapping problem by MILP, which has been observed to take too much time. This is because this technique inflates the substrate graph by introducing meta nodes and meta edges, which in turn increases the number of variables and constraints in the linear program.

In this paper, we still adopt this augmentation idea but applied it in the crossover and mutation operations of the GA. The novel approach not only keeps the integrity of the VNE problems in one stage, but also improves the mapping speed meeting online dynamic placement constraints.

B. Constraints of Embedding VNR

When a VNR arrives, the SN is supposed to decide whether such request is accepted or not based upon current remaining resources (CPU and bandwidth). If SN has enough network resources, this request will be assigned to specific substrate nodes (VNoM) and corresponding substrate paths (VLiM) with adequate resource requirements. The assignment is released and the substrate resources are returned after the request expires.

1) Constraints of VNoM

We define $\mathcal{M}(n^v)$ as the substrate node mapping from a virtual node n^v. First, each virtual node of the same VNR must be assigned to a different substrate node as shown in (1). $R^n(\mathcal{M}(n^v))$ represents the residual CPU capacity of the substrate node. Therefore, $R^n(\mathcal{M}(n^v))$ is less than or equal to the original node capacity $C(\mathcal{M}(n^v)))$. The the remaining CPU capacity of the mapped substrate node should have enough CPU capacity as described in (2). Eventually, (3) ensures that the deviation between the virtual node n^v and the selected substrate node $\mathcal{M}(n^v))$ must not exceed the radius D of the VNR. All of above three expressions must be fulfilled at the same time to make sure the successful VNoM.

$$\mathcal{M}(n^v) \neq \mathcal{M}(m^v) \tag{1}$$

$$C(n^v) \leqslant R^n(\mathcal{M}(n^v)) \tag{2}$$

$$dis(loc(n^v), loc(\mathcal{M}(n^v))) \leqslant D \tag{3}$$

where,

$$n^v, m^v \in N^v,$$
$$\mathcal{M}(n^v), \mathcal{M}(m^v) \in N^s,$$
$$R^n(\mathcal{M}(n^v)) \leqslant C(\mathcal{M}(n^v))$$

2) Constraints of VLiM

We determine a link mapping $\mathcal{M}(e^v_{nm})$ from a virtual link (from n^v to m^v) to a substrate path between two substrate nodes $\mathcal{M}(n^v)$ and $\mathcal{M}(m^v)$ respectively. Similar to $R^n(\mathcal{M}(n^v))$, $R^e(e^s)$ is the remaining bandwidth of substrate link e^s where every substrate link in $\mathcal{M}(e^v_{nm})$ should have enough bandwidth resource to allocate the virtual link as shown in (4).

$$B(e^v_{nm}) \leqslant R^e(\mathcal{M}(n^v), \mathcal{M}(m^v)) \tag{4}$$

where,

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

Only a VNR mapping, satisfying the above-stringent constraints for all nodes and links, is set as a feasible solution.

C. Performance Metrics

1) Remaining bandwidth ratio

Remaining bandwidth ratio r_b is the current residual bandwidth of a SN over the original bandwidth of all substrate links:

$$R^e(E^s) = \sum_{e^s \in E^s} (B(e^s) - \sum_{\{e^v | e^s_{uv} \in \mathcal{M}(e^v) \cup e^v \in E^v_c\}} B(e^v)) \tag{5}$$

$$r_b = \frac{R^e(E^s)}{\sum_{e^s \in E^S} B(e^s)} \tag{6}$$

E^v_c is the set of all the virtual links, which belongs to current successfully-allocated VNRs. In fact, r_b is an important performance metric to describe the embedding's efficiency. For example, when there are many VNRs rejected but the remaining bandwidth ratio remains high, which means that the substrate network has low resource utilization and it is wasting much available bandwidth resources.

2) VNR acceptance ratio

VNR acceptance ratio r_a is determined by the number of successfully mapped VNRs $a'(\tau)$ and the number of proposed VNRs $a(\tau)$ during the interval time τ. In practice, the VNR acceptance ratio, used to evaluate an algorithm's ability of batch processing, is an essential metric for the performance evaluation between the VNE algorithms.

$$r_a = \frac{a'(\tau)}{a(\tau)} \tag{7}$$

3) Objective function

In our proposed GA, we attempt to minimize the resource mapping costs as well as to balance the network loads. As our algorithm tends to map a VNR within a single stage, the objective function aims to evaluate both node and link embeddings at once, which is also called fitness function in GA terminology. Towards the loads of substrate nodes and links, we enumerate all the substrate resources (CPU capacity or bandwidth) allocated for the virtual nodes/links. It means the fitness function in this paper considers all resource usage for a VNR as a whole, as opposed to other heuristic mapping approaches [5][9], which sequentially consider the virtual nodes/links based on a ranking method. In (8), σ is a small positive constant to prevent the zero denominator. f^i_{uv} describes total amount of flows from u to v for the ith virtual link under a specific mapping scenario. And x_{mw} denotes a binary variable, which has the value 1 if the meta link is activated shown in (10); Otherwise, it is set to 0. $0 \leqslant \alpha_{uv} \leqslant R^e(e^s_{uv})$ and $0 \leqslant \beta_w \leqslant R^n(w^s)$ are parameters to control the significance of load balancing during the VNR mapping.

$$\begin{aligned} min \;\; F = & \sum_{e^s_{uv} \in E^S} \frac{\alpha_{uv}}{R^e(e^s_{uv}) - \sum_i f^i_{uv} + \sigma} \\ & + \sum_{w^s \in N^S} \frac{\beta_w}{R^n(w^s) - \sum\limits_{m^v \in N^v} x_{mw} C(m^v) + \sigma} \end{aligned} \tag{8}$$

where,

$$f^i_{uv} = B(e^v), \;\; \text{if} \;\; e^s_{uv} \in \mathcal{M}(e^v) \cup e^v \in E^v_c \tag{9}$$

$$x_{mw} = \begin{cases} 1, \;\text{if}\; \mathcal{M}(m^v) = w^s & \text{(10a)} \\ 0, \;\text{otherwise} & \text{(10b)} \end{cases}$$

Several prior solutions apply linear objective functions to facilitate the optimization process. In GA, a non-linear objective function has similar computing complexity with a linear one. The paper [21] has shown that a non-linear objective function outperforms a linear one due to the increased resource utilization. In a non-linear function, the fitness value F_{NLP} grows more quickly than using a linear function when network costs increased. Furthermore, equation (8) is intended to restrain the link mapping in case the residual bandwidth is limited. Obviously, a non-linear objective function like (8) improves the resource utilization efficiency since a small residual substrate resource may cause resource fragmentation and it is hard to be utilized for future VNRs under the unsplittable mapping.

IV. Proposed one-stage algorithm

Inspired by the process of natural evolution, a set of metaheuristics for global optimization are proposed in Evolutionary Computation (EC). GA, one of the EC techniques, has been successfully proven to produce a great performance for solving the multi-objective optimization problem with flexibility and adaptability. GA repeatedly updates a population of individual solutions. At each step, the algorithm typically selects parents from the population, and then uses them to reproduce children solutions for the next generation. The population is evolved towards an optimal solution after successive generations.

GA algorithm is made up of four operations: initialization, selection, crossover and mutation [22]. The first step is to randomly generate some possible solutions called chromosomes as the initial population. After initializing the population, the quality of each possible solution in the population is measured through the fitness function. In the selection operation, parental chromosomes are selected from the population and then produce their offspring by exchanging genes in crossover operation. Before the children chromosomes are generated and put into the population, mutation operation is performed by randomly tweaking genes to increase the diversity of the population. GA algorithm is an iterative procedure until a terminating condition has been reached. The chromosome with the best fitness value becomes the final solution.

The advantages of deploying GA in VNE problems could be summarized in three points. First, GA is a mature metaheuristic algorithm which has been proved to be effective and widely exploited in different areas. There are already some previous research [16] [17] [18] [19] applied GA in VNE related problems, which has a good performance. Second, Metaheuristic algorithms have great potential for online dynamic VNE problems since they provide near-optimal solutions as well as meeting strict time requirements. Besides, the paper [23] reveals that the GA metaheuristic has better performance than ant colony metaheuristic, which inspires us to develop a GA decision-making approach for VNE. Finally, by utilizing the exclusive features among feasible solutions, GA is such an approach to be naturally implemented in parallel [24]. The easy-composition peculiarity makes GA applicable in a distributed and parallel framework, thus further reducing the execution time.

To tailor GA in VNE problems, we refine the GA representation in a distributed parallel architecture. As mentioned above, SPs cares much on the execution time of online VNE problems. With the decreasing cost of computing devices and the rapid development of cloud technology, distributed and parallel computing has recently emerged as an effective mechanism to tackle large and complex problems. As illustrated in Fig. 3, our proposed GA can be run in several machines in a distributed manner. Except the first step refining the set of candidate nodes and the last synchronization step, the other procedures can be conducted by slave node machines in parallel. These machines operate GA algorithm independently to achieve feasible VNE embedding solutions that coordinate nodes and links in the same pace. In our framework, only the original requests with refined constraints and the mapping results are transmitted between the master and slave machines. There is no extra communication cost during the slave procedures. Therefore, the benefits of parallel working scheme is far beyond the extra cost between the master and slave machines.

In detail, each slave machine starts with the selection of an initial population. Then it deploys the crossover and mutation operators to reproduce offspring. These produced offspring are added and sorted for the next generation. If parents have better fitness values which are calculated through our objective function described in (8), their offspring have a better chance to "survive". This process keeps iterating until the number of predefined iterations have been reached and at the end, the fittest individuals are achieved.

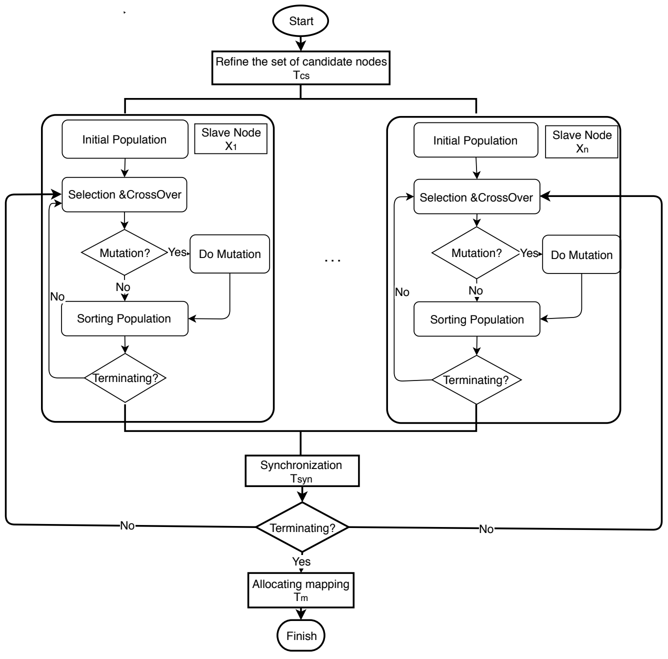

Fig. 3: Parallel execution flow chart

A. Refine the sets of candidate nodes

As described in Section III, we represent the set of all candidate nodes for a virtual node as $N^c(n^v)$. This set includes all substrate nodes within the distance limitation of the virtual node. Moreover, there are additional rigorous constraints of mapping a virtual node as shown in (1) and (2). In this step, we check all the node constraints with current available resources, and later process the candidate nodes sets. Its initiative is to avoid infeasible substrate nodes forming the population and make the following genetic productions more effective.

B. Parallel slave procedure

After the candidate nodes have been refined by node mapping constraints, the slave nodes begin the GA procedures independently. In EC, the primary step is to design a proper genetic representation for a specific problem. Consequently, we present the genetic representation and feature the genetic operations as below.

1) Genetic representation

The representation scheme determines the way of representing solutions in EC methods. GA terminology is usually analogous with natural genetics, using linear binary encoding and fixed length representations. In this paper, a candidate solution is called a chromosome which consists of several genes. In NV, a VNR can consist of several virtual links which connect virtual nodes together. The number of virtual links are variable, and the path length of each link embedding is also inconstant. Therefore, we proposed variable length representations that are more complicated than the most general case. Our proposed GA encoding method denotes each gene as a VNR link mapping solution based on an augmented graph. That is to say, a gene is started and ended by meta links and each chromosome acts as a VNR mapping solution.

Specifically, the procedure begins with a set of chromosomes which are called a population. A chromosome $\boldsymbol{c_i}$ denoted by (11) represents a feasible solution for mapping a VNR to a SN. Since $\mathcal{N}^{vl}$ indicates the number of virtual links for a request, there are totally $\mathcal{N}^{vl}$ genes in a chromosome. A gene $\boldsymbol{g_{ij}}$ is a mapping solution for the corresponding virtual link. The first subscript i indicates its chromosome whereas the second subscript j denotes the jth virtual link in the chromosome. Similarly, a node denoted by n_{ijk} can represent a substrate/meta node. The first two subscripts indicate its gene while the third one denotes its position in the gene. Moreover, a gene can be denoted by (12) with a variable length d_{ij}. The first and the last links in a gene are the meta links that represent the node mapping result 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 (13): head $\boldsymbol{H_{ijk}}$ and tail $\boldsymbol{T_{ijk}}$, where k is the index of node in the gene.

$$\boldsymbol{c_i} = \{\boldsymbol{g_{i1}}, \boldsymbol{g_{i2}}, ..., \boldsymbol{g_{ij}}, ..., \boldsymbol{g_{i\mathcal{N}^{vl}}}\} \tag{11}$$

$$\boldsymbol{g_{ij}} = \{n_{ij1}, ..., n_{ijk}, ..., n_{ijd_{ij}}\} \tag{12}$$

$$\boldsymbol{g_{ij}} = \left[\boldsymbol{H_{ijk}}, \boldsymbol{T_{ijk}}\right], \ \forall k \in (0, d_{ij}) \tag{13}$$

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

Setup an initial population is the first step in the GA process. The population $\boldsymbol{P}$ is composed of a set of chromosomes, where each chromosome is a feasible solution for embedding a VNR. There are two steps to constitute a chromosome. The first step is to randomly choose the substrate nodes 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 find a substrate path for each gene. We need to come up with some "good" potential substrate paths in the SN for mapping virtual links. Eventually, shortest paths based on hop count factor are certainly more favorable because they tend to consume fewer resources. We identify K shortest paths for each source-destination pair in the SN. Existing K shortest path algorithm (e.g. Dijkstra's algorithm) can be reliably deployed to build the path pool. The path pool is only dependent on the SN topology that can be constructed before any online VNR arrives, and it can be also reusable during the mapping procedure. Therefore, we do not count the time for this process as part of the time for our online embedding procedure.

According to the path pool, we finish the second step by uniformly selecting a path for each gene. We use uniform selection for both steps instead of another specific order because if all initial populations are chosen by a deterministic method, all parallel working machines will work with the same population, which makes parallel running meaningless.

After a chromosome is formed, we do check the feasibility of the chromosome as described in Section III-B. If the SN has enough resources to allocate all genes of the chromosome, such chromosome is regarded as a feasible solution. Generally, only the feasible chromosome is added to the population.

$$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{c_1} \\ \boldsymbol{c_2} \\ \vdots \\ \boldsymbol{c_i} \\ \vdots \\ \boldsymbol{c_M} \end{bmatrix} = \begin{bmatrix} \boldsymbol{g_{11}} & \cdots & \boldsymbol{g_{1j}} & \cdots & \boldsymbol{g_{1\mathcal{N}^{vl}}} \\ \boldsymbol{g_{21}} & \cdots & \boldsymbol{g_{2j}} & \cdots & \boldsymbol{g_{2\mathcal{N}^{vl}}} \\ \vdots & \ddots & & \ddots & \vdots \\ \boldsymbol{g_{i1}} & \cdots & \boldsymbol{g_{ij}} & \cdots & \boldsymbol{g_{i\mathcal{N}^{vl}}} \\ \vdots & \ddots & & \ddots & \vdots \\ \boldsymbol{g_{M1}} & \cdots & \boldsymbol{g_{Mj}} & \cdots & \boldsymbol{g_{M\mathcal{N}^{vl}}} \end{bmatrix} \tag{14}$$

Otherwise, if the chromosome candidate does not pass the feasibility measurement, we will go back to step 1 to select and check another candidate chromosome again. This process continues until a feasible chromosome is selected. In some special cases, the randomly initialization process could not find a feasible chromosome due to the exhausted available resources. Instead of rejecting the VNR directly, we select and stamp some infeasible chromosomes into the population. Therefore, the VNR still have chance to enter crossover and mutation operations to produce feasible children chromosomes. After initialization, the population $\boldsymbol{P}$ in Equation (14) basically has M chromosomes, and each chromosome has $\mathcal{N}^{vl}$ virtual links. Thus, the size of $\boldsymbol{P}$ is $M \times \mathcal{N}^{vl}$.

3) Selection and Crossover

Selection operation is conducted to choose parent chromosomes from the initial population for the crossover operation. In general, one or several pairs of parent chromosomes can be chosen from this step. Each selected parent pair then performs crossover operation. In this paper, to improve the degree of parallelism, we only choose one-pair parents. We arrange the selection scheme based on random selection with replacement as paper [19]. The replacement means the parent chromosomes will be put back into the population after crossover.

The crossover operation based on genes operates each gene with its corresponding gene in the other parent chromosome. Let take two selected parent chromosomes are $\boldsymbol{c_s}$ and $\boldsymbol{c_r}$ as an example. We use $\boldsymbol{c_{(M+1)}}$ and $\boldsymbol{c_{(M+2)}}$ to denote the two new children chromosomes generated from the parent chromosomes. The genes inside parent chromosomes can be denoted as $\boldsymbol{g_{sj}}$ and $\boldsymbol{g_{rj}}$, where j indicates the jth virtual link of a request. Obviously $\boldsymbol{g_{sj}}$ and $\boldsymbol{g_{rj}}$ have the same starting and ending meta nodes. Each chromosome should exchange partial genes with its counterpart through a crossover point. For each pair of genes, we first identify a common node. 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 indices of source or destination node, we denote the node as a common node. As a result, there are two scenarios in the crossover operation:

a) : There are more than one common node in parental genes. Thus, one common node is selected to become the crossover point. With the common node known as the demarcation point, we swap the second parts of two corresponding genes to generate two children genes. Apparently, such children generated in this fashion are still valid paths. The children genes (15) and (16) are defined as below:

$$\boldsymbol{g_{(M+1)j}} = \left[\boldsymbol{H_{sju}}, \boldsymbol{T_{rjv}}\right] \tag{15}$$

$$\boldsymbol{g_{(M+2)j}} = \left[\boldsymbol{H_{rjv}}, \boldsymbol{T_{sju}}\right] \tag{16}$$

b) : There is no common node between two-parent genes. In this case, a link is selected as the crossover point in each parent gene. To make sure the children's genes are still a valid path, a partial path obtained from the shortest path pool connects the children genes. For instance, we randomly select link $(n_{sju}, n_{sj(u+1)})$ as a crossover point for $\boldsymbol{g_{sj}}$, and then select $(n_{rjv}, n_{rj(v+1)})$ for $\boldsymbol{g_{rj}}$. A substrate path between n_{sju} and $n_{rj(v+1)}$ is chosen from the source-destination path set $P^s(n_{sju}, n_{rj(v+1)})$. Similarly, a substrate path is picked from $P^s(n_{rjv}, n_{sj(u+1)})$ to connect the partial child genes. The results of crossover operator without the common node should be (17) and (18).

$$\boldsymbol{g_{(M+1)j}} = \left[\boldsymbol{H_{sju}}, p_{n_{sju}, n_{rj(v+1)}}, \boldsymbol{T_{rj(v+1)}}\right] \tag{17}$$

$$\boldsymbol{g_{(M+2)j}} = \left[\boldsymbol{H_{rjv}}, p_{n_{rjv}, n_{sj(u+1)}}, \boldsymbol{T_{sj(u+1)}}\right] \tag{18}$$

After crossover, the child gene may contain loops, which is an invalid path. Hence, each gene will have the validation check to remove loops. Subsequently, we build the children chromosomes from children genes through previous crossover operations. Previous research [19] only performs the crossover operation in the link mapping stage, and there are $2^{N^{vl}}$ combinations to form a chromosome. In the proposed algorithm, the node mapping solution is indicated by the meta links in the genes, which may be messed after the exchange operation in the crossover. Specifically, when the crossover operation is conducted, the parent genes will swap partial nodes and generate the children genes. This operation will intermingle the node mapping combinations due to the meta links exchanged between parent genes. Therefore, we have to carefully organize the children genes and a virtual node mapped into the same substrate node in different genes of a child chromosome. Indeed, there is only one way to compose the children chromosomes based on the node mapping results in each gene crossover.

The composition of a chromosome becomes a matching problem [25] in graph theory. It means that, for a given VNR, a matching is a set of pairwise non-adjacent virtual links, 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, and every link connects a node from a set to one in the other set. 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, the VN is a bipartite graph. We select the meta links in parent chromosomes as demonstrated in Fig. 4, after crossover the genes $A-B$ and $B-C$, the children genes could be integrated into children chromosomes without node mapping disorder.

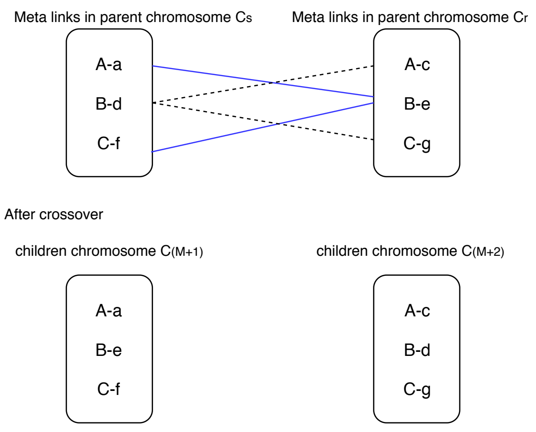
Meta links in parent chromosome Cs
Meta links in parent chromosome Cr
A-a
B-d
C-f
A-c
B-e
C-g
After crossover
children chromosome C(M+1)
children chromosome C(M+2)
A-a
B-e
C-f
A-c
B-d
C-g

Fig. 4: Node mapping results after crossover for a bipartite graph

However, the VNR 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. As shown in the blue solid line of Fig. 5, when crossovering the gene $A-C$, the meta links in the child gene will lead to the node mapping of virtual node A to substrate node c, which is contrary to the result of crossover of the gene $A-B$ where virtual node A is mapped to a. To deal with this issue, we identify some virtual links in the VN as free crossover links and others to be restricted crossover links. The restricted crossover links limit the meta link exchange in the crossover operations to avoid node mapping disorder. Additionally, we prefer as many free crossover links as possible in a non-bipartite VN, thereby increasing the crossover freedom for the gene production. To recognize and maximize the free crossover links in a non-bipartite VN, the problem could be modeled as a maximum cut problem [26]. The maximum cut problem is to find the largest bipartite subgraph of the current graph. It has been proved to be a $\mathcal{NP}$-complete problem.

Therefore, we determine the free crossover links (eg. $A-B$ and $B-C$) and the restricted crossover links (eg. $A-C$) by identifying the largest bipartite subgraph in the VN. It is noted that the crossover operations happens in free crossover links. To compose a chromosome, after coloring the bipartite subgraph, the restricted crossover links are appended according to the node mapping results.

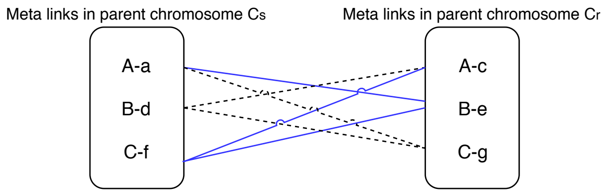
Meta links in parent chromosome Cs
Meta links in parent chromosome Cr
A-a
B-d
C-f
A-c
B-e
C-g

Fig. 5: Node mapping results disordered after crossover

4) Mutation

Mutation operation aims to expand the solution space and broaden the search, thereby avoiding local optima. In this paper, the mutation operation is relied on genes as the same the crossover operation. Each gene in the children chromosomes $\boldsymbol{c}_{(M+1)}$ and $\boldsymbol{c}_{(M+2)}$ goes through the mutation operation with a fixed probability called mutation rate. The traditional method is to choose two nodes called mutation points in a gene and the partial path between these two nodes will be replaced by an alternative path from our path pool [19]. In our proposed algorithm, a gene based on the augmented graph represents both virtual node and link mapping results. If the node mapping result is mutated, all genes in a children chromosome have to be updated. The mutation procedure is shown in Algorithm 1.

The first step is to check if a meta link is included between the mutated nodes, which means one of the mutation points is a meta node. In addition, a meta link represents the mapping of a virtual node to a substrate node; as a result, we have to update the node mapping over the whole chromosome if meta link mutation occurs. To mutate a node mapping of n^v, an alternative nodes m^s from our refined candidate node set $N^c(n^v)$ is selected. Consequently, m^s will replace the current mapping node n^s. We evaluate the new node m^s and the non-meta mutated node as the updated mutation points. Finding all other genes including the node $\mu(n^v)$ and updating all corresponding meta links by the new substrate node mapping result are a must. As we know that 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. The new node m^s is an intermediate node which connects to a meta node and a substrate node.

Accordingly, we replace the new node m^s as the updated mutation point for the next step. Then the mutation operation goes to the second step: path mutation. An alternative path chosen from the path pool substitutes current partial route and connects two updated mutation points.

Algorithm 1 Mutation Operation

```
 1: procedure MUTATION
 2:     for a gene in a chromosome do
 3:         Generating mutation points
 4:         Selecting two nodes in the gene as mutation points
 5:         if a selected node is a meta node $\mu(n^v)$
 6:             goto node mapping mutation
 7:         else:
 8:             goto path mapping mutation
 9: procedure NODE MAPPING MUTATION
10:     Selecting an alternative node $m^s$ from $N^c(n^v)$
11:     make $\mathcal{M}(n^v)$= $m^s$
12:     for all genes contain $\mu(n^v)$ do
13:         Replace $\mathcal{M}(n^v)$ as the mutation point
14:         goto path mapping mutation
15: procedure PATH MAPPING MUTATION
16:     choosing an alternative path between two mutation
        points from path pool
```

5) Sorting population

After crossover and mutation operations, two children chromosomes are created. Then, it is the time to update the population for the next generation. The new chromosomes are sorted by their fitness values together with other chromosomes. Only the best M (population size) chromosomes are saved as in the updated population. A new iteration starts with the updated population that goes to the selection procedure to figure out parents for another crossover and mutation forming the succeeding generation. This procedure will be ultimately stopped when the maximum count is reached or there are no different children chromosomes available.

C. Synchronization and Allocation

The synchronization starts when all slave working nodes have terminated their GA procedures. The best chromosomes from each slave working nodes are synchronized and could become the initial population for the next round slave procedure. Finally, the best chromosome after synchronization is the final VNE solution. The last step is to allocate the VNR, then the residual resources of the SN get updated.

D. Execution time analysis

In terms of execution time in online VNE problems, SPs expects to achieve rapid and efficient online embedding. By utilizing the parallel distributed framework, the execution time can be extremely reduced with sufficient computing resources. In this paper, we define the number of slave working procedures as the parallel level. The parallel level can be tuned according to the trade-off between available computing resources and the expected performance.

To analyse how much time could be saved in the proposed distributed parallel architecture, we firstly need to consider the time spending without a parallel structure. It means 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.

In parallel running paradigm as shown in Fig. 3, there are two procedures running sequentially: refining the set of candidate nodes (T_{cs}), synchronization (T_{syn}), and allocating VNR(T_m). These procedures, through simulation, only accounts for 0.9% of the total parallel execution time on average as depicted in Fig. 10. Therefore, we approximate the total parallel execution time as the execution time of parallel running procedures. In terms of parallel running procedures, the execution time is equal to the maximum execution time amongst all parallel nodes because the synchronization procedure should wait until the slowest parallel node finishes its task.

Furthermore, we evaluate the upper bound of the mean value of the total parallel execution time according to Cramer-Chernoff method and Jensen's inequality. Finally, we found that the parallel running scheme can improve the execution time from linear time to logarithmic time. The detailed derivation can be found in [19].

TABLE II: Compared Algorithms

GAOne	Our proposed parallel GA for one-stage mapping.
PBGA	Greedy worst fit node mapping parallel Path Based GA Link Mapping.
SBGA	Greedy worst fit node mapping parallel Segment Based GA Link Mapping.
G-SP	Greedy worst fit node mapping Shortest path based link mapping.
R-ViNE	Randomized node mapping unsplittable link mapping based on shortest path.
D-ViNE	Deterministic node mapping unsplittable link mapping based on shortest path.

V. Performance evaluation

This section presents the simulation settings followed by the performance results. We select the rival algorithms with the same evaluation environment to make the comparison fairly. These algorithms are listed in Table II along with our proposed Genetic-based algorithms. The compared algorithms we chose are based on the performance-oriented and speed-oriented aspects.

Specifically, we selected DViNE and RViNE in [5] since [5] is the most popular VNE research work and also considered as a benchmark in several research papers, even in recent research. Besides, these two algorithms formulate the VNE as MCF-based model using the augmented graph. Moreover, SP [9] is considered for comparison because it uses the shortest path algorithm for link mapping stage, which is widely deployed by a numerous number of heuristic algorithms as mentioned above, and also because it is known as the fastest algorithm due to its simplicity. In addition, we compared our proposed algorithm with [19] that merely employs a GA in link mapping stage. Comparing with [19], we can clearly learn about how much our one-stage algorithm improves the performance.

A. Simulation settings

We set up the simulation parameters with the same settings in [5] and [19]. We randomly generate three different SN typologies with average 50 nodes in 25×25 grids by Waxman algorithm. All the simulation results are gotten from the average based on these SNs. CPU capacity and bandwidth resources of the SNs are real numbers uniformly distributed between 50 and 100. We assume that the VNRs dynamically arrive in a Poisson distribution with average rates ranging from 4 to 8 requests per 100 time units, and each has an exponentially distributed lifetime with an average of 1000 time units. Each simulation was run for 50000 time units that are 50 times longer than the average lifetime of a request. Basically, we have 6 simulations for each SN. Each VNR consists of various numbers of virtual nodes and virtual links. The number of virtual nodes follows a uniform distribution between 2 and 10. CPU capacity requirements of virtual nodes are uniformly distributed from 0 to 20 whereas the bandwidth requirements of the virtual links are uniformly distributed from 0 to 50.

B. Performance metrics

- Acceptance ratio r_a: the percentage of the number of accepted VNRs calculated in (7).
- Execution time: the average time consumed for allocating a VNR. All algorithms are executed on Ubuntu 18.10 64-bit with 7.7GiB memory and Intel Core i5-6200U CPU@2.30GHz$\times$4, and the linear program solver used in D-ViNE and R-ViNE is glpk same as [5].
- Remaining bandwidth ratio r_b: the sum of bandwidth of all available substrate links as shown in (6).

C. Evaluation Results

The performance metrics in this article include average acceptance ratio, remaining bandwidth ratio, embedded path length and the execution time analysis. We pay more attention on link utilization analysis than node utilization in simulation, since we assume the node mapping is unsplittable. The acceptance ratio in Fig. 6 can somehow indicate the node utilization. Generally, the link mapping solution contains multiple substrate paths, thereby deserving more attentions in this paper. Fig. 6, 7, 8 describe the average values over arrival rates from 4 to 8 per 100 time units with 95% CI (confidence interval).

The execution time is measured for each procedure as shown in the Fig. 3. Procedures running in the master node have to be executed in sequential while the parallel procedures can be independently performed. Hence, the execution time of the parallel procedures depends on the maximum execution time amongst all parallel working machines since the parallel algorithm should wait until the slowest parallel machine accomplishes its task. The time complexity and convergence analysis of a distributed parallel architecture had been discussed in paper [19]. In a nutshell, the parallel scheme can be finished in logarithmic time. According to the convergence analysis [19], the acceptance ratio reaches converged values when the parallel level goes up to 16. Definitely, the parallel level can be adjusted freely regarding a time restriction or a highly expected acceptance requirement. We set the parallel level (the number of slave parallel machines) as 16 in the simulation.

1) VNR Acceptance Ratio

Fig. 6 shows the average VNR acceptance ratio as a function of the VNR arrival rate. As cited, VNR acceptance ratio is one of the most vital metric to evaluate different VNE algorithms. In Fig. 6, the acceptance ratios of all selected algorithms decay with the increase of VNR arrival rates. It is because the substrate network is getting congested and it does not have enough resources for embedding the VNRs.

Besides, we can find our proposed GAOne algorithm outperforms other selected algorithms over all arrival rates. The differences between the best compared algorithm and GAOne is more than 10%. This result indicates that GAOne is an efficient and effective VNE algorithm. This observation also implies that our GAOne with a comprehensive objective function can generate more efficient solutions. Moreover, the results verify the benefits of one-stage strategy.

2) Average remaining bandwidth and average path length

The average remaining bandwidth ratio is plotted in Fig. 7. Similar with VNR acceptance ratio, the average remaining bandwidth ratio of every selected algorithms decreases along with the increase of arrival rates.

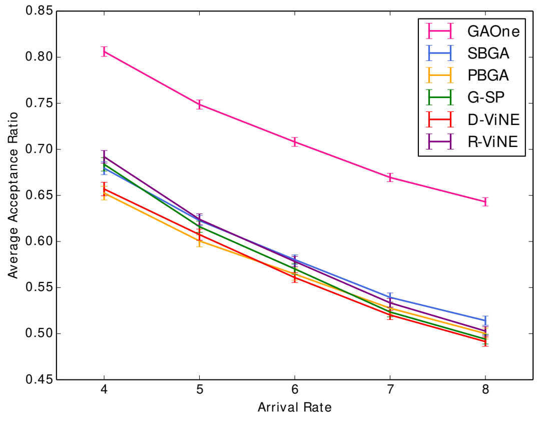

Fig. 6: Average acceptance ratio over arrival rates

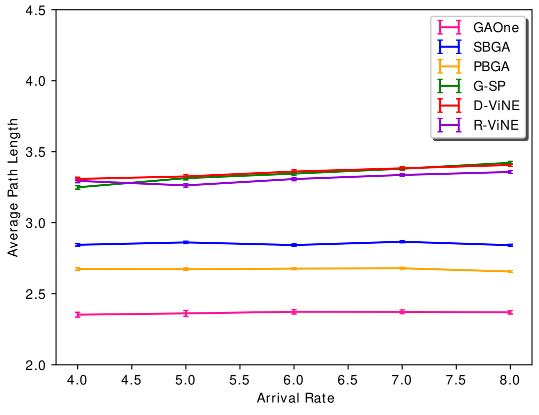

Fig. 8: Average path length analysis

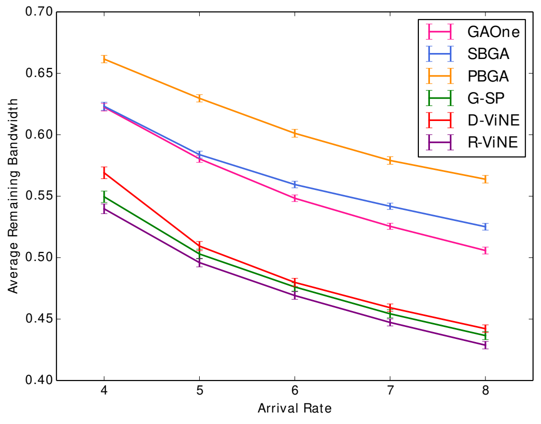

Fig. 7: Average remaining bandwidth over arrival rates

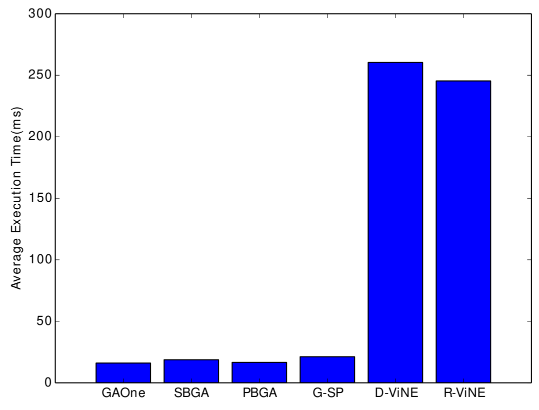

Fig. 9: Total execution time over different algorithms

Combined with the results in Fig. 6, the greedy solutions (G-SP, D-ViNE, R-ViNE) occupy more network resources in Fig.7, while the load-balanced solutions (GAone, SBGA, PBGA) utilize the bandwidth more efficiently. We can observe that the greedy solutions consume more substrate link resources when embedding a virtual link. This recognition where the load-balanced algorithms save much more the bandwidth resources for future VNRs saving derives from less average path lengths shown in Fig.8.

Moreover, amongst the load-balanced solutions, our proposed GAOne takes the least path length and has smaller remaining bandwidth ratio. The efficiency of our proposed GAOne leads to accepting more VNRs as shown in Fig. 6. This observation depicts the higher bandwidth utilization of GAone compared to SBGA and PBGA algorithms.

3) Time complexity

As shown in Fig. 9, D-ViNE and R-ViNE need much more time to approach a VNE solution due to their relaxed linear programming mechanism. In particular, D-ViNE consumes more time than R-ViNE due to its deterministic variation. The G-SP performs the greedy node mapping and greedy shortest path link mapping, which simplifies the problem with small consuming time. However, G-SP still takes longer execution time than the solutions in our distributed parallel framework.

With regard to the parallel algorithms, SBGA and PBGA essentially realize a partial parallelism as they separate node and link mapping and run the node mapping stage sequentially. The average execution time is 18.74ms and 16.62ms for SBGA and PBGA respectively. SBGA consumes more time because it considers the path restructuring while PBGA is a path based algorithm. Our proposed algorithm GAOne considers the path restructuring as well as the node remapping and takes 16.05ms to allocate one request on average. The gain of execution time of GAOne is from the fully parallel running mechanism of one-stage strategy.

Moreover, as exposed in Fig. 10, four procedures with average execution time are simulated in our proposed GAOne algorithm. Parallel running procedures T_x takes the most time whilst other sequential functions relatively take less time compared with the parallel working procedure. The parallel procedure takes 99% of the total time, which implies our proposed framework realises a full one-stage based on the distributed parallel scheme.

VI. Conclusion

This paper presents a one-stage VNE solution which has merely received a few research investigations due to its

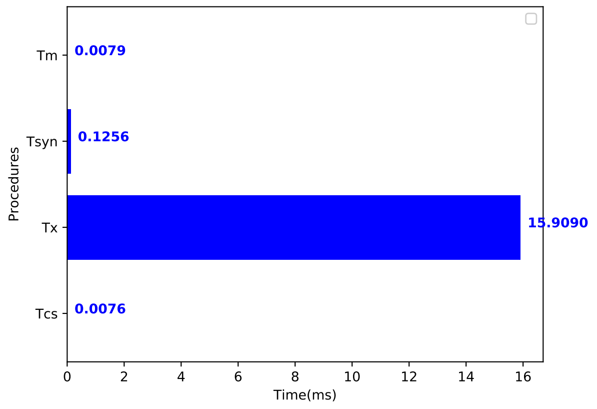

Fig. 10: Execution time on different procedures

complexity. Our proposed solution utilizes a novel GA in an augmented graph combined with Graph Theory to solve the VNE problem without separated stages. Comprehensive simulations are conducted to prove the effectiveness of the proposed algorithm. Performance results show that the GAOne algorithm outperforms typical heuristic algorithms. With respect to average VNR acceptance ratio, the proposed GAOne exceeds other selected algorithms at least 10%. In terms of execution time, GAOne algorithm reduces the time complexity to logarithmic time through a distributed parallel paradigm. The simulation demonstrates that our proposal consumes much less execution time than all other rivals. In addition, the results confirm our argument that the one-stage mapping greatly broadens the solution space and provides more efficient mapping results.

References

[1] Juliver Gil Herrera and Juan Felipe Botero. Resource allocation in nfv: A comprehensive survey. *IEEE Transactions on Network and Service Management*, 13(3):518–532, 2016.

[2] Jorge Carapinha, Peter Feil, Paul Weissmann, Saemundur E Thorsteinsson, Çağrı Etemoğlu, Ólafur Ingþórsson, Selami Çiftçi, and Márcio Melo. Network virtualization-opportunities and challenges for operators. In *Future Internet Symposium*, pages 138–147. Springer, 2010.

[3] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck, and Raouf Boutaba. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 18(1):236–262, 2015.

[4] H. Cao, H. Hu, Z. Qu, and L. Yang. Heuristic solutions of virtual network embedding: A survey. *China Communications*, 15(3):186–219, 2018.

[5] 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.

[6] 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.

[7] H. Cao, S. Wu, Y. Guo, H. Zhu, and L. Yang. Mapping strategy for virtual networks in one stage. *IET Communications*, 13(14), 2019.

[8] H. Cao, Y. Zhu, G. Zheng, and L. Yang. A novel optimal mapping algorithm with less computational complexity for virtual network embedding. *IEEE Transactions on Network and Service Management*, 15(1):356–371, 2018.

[9] 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.

[10] P. Zhang, H. Yao, C. Qiu, and Y. Liu. Virtual network embedding using node multiple metrics based on simplified electre method. *IEEE Access*, 6:37314–37327, 2018.

[11] X. Cheng, S. Su, Z.Zhang, H. Wang, F.Yang, Y. Luo, and J. Wang. Virtual network embedding through topology-aware node ranking. *ACM SIGCOMM Computer Communication Review*, 41(2):38–47, 2011.

[12] Haotong Cao, Yongxu Zhu, Longxiang Yang, and Gan Zheng. A efficient mapping algorithm with novel node-ranking approach for embedding virtual networks. *IEEE Access*, 5:22054–22066, 2017.

[13] Shuxian Jia, Guiyuan Jiang, Peilan He, and Jigang Wu. Efficient algorithm for energy-aware virtual network embedding. *Tsinghua Science and Technology*, 21(4):407–414, 2016.

[14] Zhongbao Zhang, Xiang Cheng, Sen Su, Yiwen Wang, Kai Shuang, and Yan Luo. A unified enhanced particle swarm optimization-based virtual network embedding algorithm. *International Journal of Communication Systems*, 26(8):1054–1073, 2013.

[15] Hao Di, Hongfang Yu, Vishal Anand, Lemin Li, Gang Sun, and Binhong Dong. Efficient online virtual network mapping using resource evaluation. *Journal of Network and Systems Management*, 20(4):468–488, 2012.

[16] 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.

[17] X. Mi, X. Chang, J. Liu, L. Sun, and B. Xing. Embedding virtual infrastructure based on genetic algorithm. In *13th International Conference on PDCAT*, pages 239–244, 2012.

[18] 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.

[19] Qiao Lu, Khoa Nguyen, and Changcheng Huang. Distributed parallel algorithms for online virtual network embedding applications. *International Journal of Communication Systems*, page e4325, 2020.

[20] C. Huang and J. Zhu. Modeling service applications for optimal parallel embedding. *IEEE Transactions on Cloud Computing*, 2018.

[21] Q. Lu and C. Huang. Distributed parallel vn embedding based on genetic algorithm. In *2019 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6, 2019.

[22] M. Melanie. *An introduction to genetic algorithms*. MIT press, 1998.

[23] 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.

[24] H. Mühlenbein. Parallel genetic algorithms in combinatorial optimization. In *Computer science and operations research*. Elsevier, 1992.

[25] L. Lovász and M. Plummer. *Matching theory*. American Math. Soc.

[26] Charles Delorme and Svatopluk Poljak. Laplacian eigenvalues and the maximum cut problem. *Mathematical Programming*, 62(1-3):557–574, 1993.

Qiao Lu received her B.Eng from Donghua University and M.Sc from Nanyang Technological University in Telecommunications Engineering. Currently, she is working toward the PhD degree at the Department of Systems and Computer Engineering, Carleton University. Her research interests include Network Function Virtualization, Software Defined Network, Virtual Network Embedding optimization, particularly with artificial intelligence.

Khoa TD Nguyen received his M.Sc. degree in Telecommunications Engineering from the University of Sunderland, England, in 2013. He is currently pursuing the Ph.D. degree in Electrical and Computer Engineering at the Department of Systems and Computer Engineering, Carleton University, Canada. His main research interests are in communication networks, network virtualization, and machine learning

Changcheng Huang received his B. Eng. in 1985 and M.Eng. in 1988 both in Electronic Engineering from Tsinghua University, Beijing, China. He received a Ph.D. degree in Electrical Engineering from Carleton University, Ottawa, Canada in 1997. From 1996 to 1998, he worked for Nortel Networks, Ottawa, Canada where he was a systems engineering specialist. He was a systems engineer and network architect in the Optical Networking Group of Tellabs, Illinois, USA during the period of 1998 to 2000. Since July 2000, he has been with the Department of Systems and Computer Engineering at Carleton University, Ottawa, Canada where he is currently a full professor. Dr. Huang won the CFI new opportunity award for building an optical network laboratory in 2001. He is an associate editor of Springer Photonic Network Communications. Dr. Huang is a senior member of IEEE.