GAMap: A Genetic Algorithm-based Effective Virtual Data Center Re-Embedding Strategy

Anurag Satpathy, *Graduate Student Member, IEEE,* Manmath Narayan Sahoo, *Member, IEEE,* Chittaranjan Swain, *Graduate Student Member, IEEE,* Muhammad Bilal, *Senior Member, IEEE*, Sambit Bakshi, *Senior Member, IEEE*, and Houbing Song, *Fellow, IEEE*

***Abstract*—Network virtualization allows the service providers (SPs) to divide the substrate resources into isolated entities called virtual data centers (VDCs). Typically, a VDC comprises multiple cooperative virtual machines (VMs) and virtual links (VLs) capturing their communication relationships. The SPs often re-embed VDCs entirely or partially to meet dynamic resource demands, balance the load, and perform routine maintenance activities. This paper proposes a genetic algorithm (GA)-based effective VDC re-embedding (*GAMap*) framework that focuses on a use case where the SPs relocate the VDCs to meet their excess resource demands, introducing the following challenges. Firstly, it encompasses the re-embedding of VMs. Secondly, VL re-embedding follows the re-embedding of the VMs, which adds to the complexity. Thirdly, VM and VL re-embedding are computationally intractable problems and are proven to be $\mathcal{NP}$-Hard. Given these challenges, we adopt the GA-based solution that generates an efficient re-embedding plan with minimum costs. Experimental evaluations confirm that the proposed scheme shows promising performance by achieving an 11.94% reduction in the re-embedding cost compared to the baselines.**

***Index Terms*—Virtual Data Centers, Resource Management, Data Centers, Genetic Algorithm, Re-embedding.**

I. Introduction

Network virtualization has enabled the service providers (SPs) to divide the substrate resources into independent executable entities called virtual data centers (VDCs) [1]. Such a partitioning scheme assists the SPs in achieving service isolation and effectively utilizing the substrate network resources [2], [3]. Typically, a VDC consists of multiple cooperative virtual machines (VMs) [4]. These VMs are dependent, and their dependencies are captured via virtual links (VLs). The VMs and VLs have different requirements for various kinds of resources pictorially depicted in Figure 1. For instance, the demand $4/4$ of a VM $v_{1,2}$ states that its requirement comprises 4 core CPU and 4 GB of RAM. On the other hand, the numeral 0.6 on the VL interconnecting $v_{1,1}$ and $v_{1,2}$ denotes its bandwidth demand (in $Gbps$). From a SPs perspective, assigning resources to VDCs is particularly challenging owing to the inherent dependencies among the VMs. The process of allocating substrate resources to VDCs, referred to as virtual data center embedding (VDCE), is a *computationally intractable* problem [5].

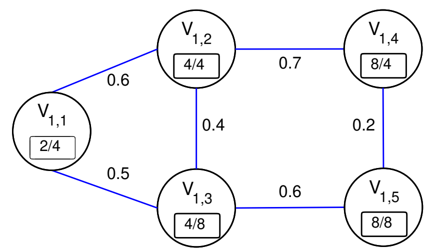


Fig. 1: A VDC with corresponding resource demands [5].

After the initial embedding, a VDC may require relocations to balance the load, perform maintenance activities, meet demand spikes, etc. [2]. In this work, we restrict our attention to handling the complete relocation of dynamic VDCs experiencing fluctuating resource demands. A dynamic VDC encompasses various stages in its life span, as reflected in Figure 2. In this regard, re-embedding the VDCs is quite challenging for the SPs for the following reasons. Firstly, re-embedding both VMs and VLs is a complicated operation. Secondly, the remapping should ensure that requested resources with minimum remapping costs are allocated. The requirements mentioned above render the relocation of VDCs a non-trivial operation. Literature on VDCE [2], [3] is focused on generating immutable assignments for static VDCs. Considering dynamic VDCs, Guo *et al.* [6] proposed a heuristic to perform temperature-aware dynamic embedding of VDCs to reduce the hot spots. On the other hand, Nam *et al.* [7], [8] discussed dynamic server consolidation strategies to reduce energy consumption in hosting multiple VDCs. To simultaneously minimize the scheduling delay and energy expended, Zhani *et al.* [9] presented a dynamic embedding scheme based on VM migration. Some other works focused on reducing the migration overheads for dynamic VDCs are discussed in [10], [11]. However, little impetus is given to lowering the remapping costs for embedding dynamic VDCs. In this context, a naive relocation strategy can deleteriously impact the SPs' acceptance ratio and revenue in the long run.

Manuscript received November 1, 2022; Accepted: XXX, Published: XXXX. This paper was recommended by Associate Editor XYZ. (Corresponding authors: Manmath Narayan Sahoo)

A. Satpathy is with the Department of Computer Science at Missouri University of Science and Technology, MO, USA (e-mail: anurag.satpathy@mst.edu)

M.N. Sahoo, C. Swain, and S. Bakshi are with the Department of Computer Science and Engineering, National Institute of Technology, Rourkela, Odisha 769008, India (e-mail: anurag.satpathy@ieee.org; sahoom@nitrkl.ac.in; chittaranjanswain518@gmail.com; bakshisambit@ieee.org).

M. Bilal is with the School of Computing and Communications, Lancaster University, Lancaster, United Kingdom (e-mail: m.bilal@ieee.org).

H. Song is with the Department of Information Systems, University of Maryland, Baltimore County (UMBC), Baltimore, MD 21250 USA, (email: h.song@ieee.org)

This article has been accepted for publication in IEEE Transactions on Green Communications and Networking. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TGCN.2023.3345542

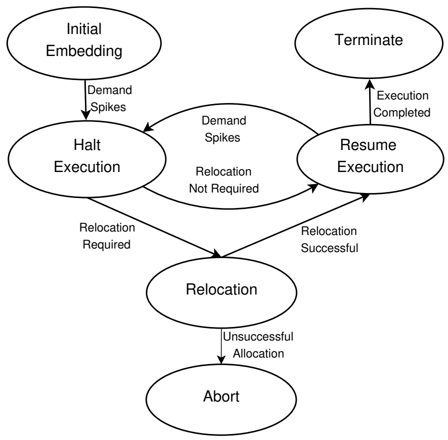


Fig. 2: A VDC Life-Cycle State Diagram.

Hence, this work proposes a model called genetic algorithm-based effective VDC re-embedding (*GAMap*), which aims to reduce the remapping costs in the complete relocation of dynamic VDCs. We propose a framework *GAMap* that adopts a genetic algorithm (GA) based solution strategy to generate an efficient relocation plan for remapping multiple VDCs with minimum remapping costs. The motivation for adopting GA as a solution strategy are: (*i.*) ease of implementation, (*ii*) is population-based, (*iii.*) uses probabilistic transition rules, (*iv.*) is robust, and (*v.*) works on different population representations [12]. Specifically, robustness is imparted to the solution owing to its ability to handle noisy and imperfect data and is often unfazed by perturbations in the search space. Moreover, to improve the solution quality, we have adopted an improved crossover [10] followed by mutations. The key **contributions** of this work are summarized below.

- We discuss a framework *GAMap* focusing on generating re-embeddings for VDCs experiencing resource expansion at minimum embedding cost (**Section III**).
- As already discussed in Section I, VDC re-embedding is $\mathcal{NP}$-Hard [13] implying an obvious trade-off between *solution quality* and *computational time* for increasing problem size. Exact approaches focus on the former, whereas heuristic approaches focus on the latter. Meta-heuristics strike a trade-off between these metrics. Therefore, we adopt GA to solve the above-discussed problem. The re-embedding is generated using an improved crossover with a unique encoding wherein a chromosome depicts the re-embedding of a VDC, subsequently followed by bit-flop mutations (**Sections IV and V**).
- To validate the performance of *GAMap*, its performance is compared with two baselines: Virtual Network Embedding based on Genetic Algorithm (VNE-GA) [14] and a Greedy-Heuristic. Further analysis ascertains that *GAMap* achieves an 11.94% reduction in the re-embedding cost compared to the baselines (**Section VI**).

The remaining paper is organized as follows. **Section II** presents an in-depth literature discussion. An insight into the components of *GAMap* is provided in **Section III**. The modeling of the VDC re-embedding is shown in **Section IV**. The GA-based solution is explained in **Section V**. The simulation setup and the analysis of the results are elaborated in **Section VI**. In **Section VII**, we conclude the work and highlight the scope for future research.

II. Literature Survey

This literature review on VDCE from the perspective of *dynamic deployment* scenarios where the assignment and/or demands of the VDCs are changeable over time is presented in this section. The literature primarily focuses on handling static VDC deployment, whereas some attempts to address dynamic VDCs can be found in [6]–[8], [10]. In this regard, the works in [6] presented a temperature-aware VDC embedding strategy that minimizes the number of hot spots by reducing the maximum hot air drawn to each server rack. Alternatively, Nam *et al.* [7], [8] discussed two different strategies to reduce energy consumption in hosting multiple VDCs. The former focused on improving the *resource efficiency* and *energy efficiency*, whereas the latter presented server consolidation strategies to embed/re-embed dynamic VDCs.

Some resource relocation policies based on VM migration have been explored in [9], [11], [15]–[17]. In this regard, the works in [15]–[17] have concentrated on developing scheduling policies for migrating correlated VMs. In [17], authors presented serial and parallel migration policies to relocate VMs across servers. To further improve the performance, the authors in [16] proposed an improved serial migration strategy for migrating correlated VMs. As service downtime is an essential performance indicator of any migration strategy, a modified serial migration strategy that outperforms the above-discussed policies is discussed in [15]. On the other hand, relocation strategies specific to VDCs are considered in [9], [11]. A dynamic embedding strategy based on VM migration to simultaneously reduce the scheduling delay and the energy consumption of VDCs is considered in [9]. Alternatively, Bari *et al.* [11] discussed an interleaved scheduling algorithm for migrating VMs of VDCs across a substrate network. The authors targeted maximizing the number of parallel migrations by grouping VMs into resource-independent groups, minimizing migration and service downtime.

From the above discussions, it can be safely interpreted that researchers have extensively investigated VDCE. However, dynamic deployment techniques primarily focus on reducing energy consumption and migration overheads and often ignore the re-embedding cost, which is of utmost importance for a SP [2]. In this regard, the closest work is presented in [10] and presents a re-embedding policy using GA; however, *GAMap* differs in the following aspects. Firstly, the work in [10] focuses on a limited use case, wherein selective re-embedding of solution components (SCs) is performed. Alternatively, *GAMap* focuses on a broader use case of re-embedding an entire VDC, which is more complex. Secondly, the overall objective of [10] was to balance the migration and re-embedding cost of migrating SCs. However, some recent research in this area [5] has emphasized that the reduction of the embedding/re-embedding costs is more critical compared to migration overheads, for which sophisticated solutions, as discussed in [11], are already in place. Therefore, considering migration and re-embedding costs in [10] dilates its

This article has been accepted for publication in IEEE Transactions on Green Communications and Networking. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TGCN.2023.3345542

performance, specifically from the perspective of reduced re-embedding cost. Thirdly, encoding and initial population generation (**Section V-B1**), fitness computation (**Section IV-B3**), and mutation (**Section V-B2**) operators are vividly different in the following ways. Concerning the encoding, the work in *GAMap* models a gene entry as a SC, whereas in *GAMap*, we consider it a server index hosting a VM. On the other hand, as opposed to random solutions in the initial population, we adopt a more pragmatic way of constructing the initial population comprising a mix of greedy and random solutions to balance exploration and exploitation. The fitness computation of *GAMap* and [10] are different as the former focuses solely on reducing the re-embedding cost, whereas the latter derives it considering migration and re-embedding cost of SCs. Finally, we adopt a bit-flop mutation as exchanging sever indices, whereas in [10] redundant links are replaced with 0. Therefore, this work proposes a SP-centric framework *GAMap* for complete VDC re-embedding with minimum cost.

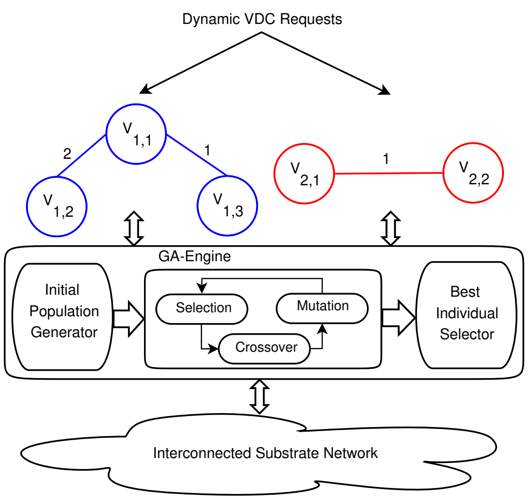


Fig. 3: Architecture of *GAMap*.

III. Architecture of GAMap

The overall architecture of *GAMap* is illustrated in Figure 3. It takes as input the re-embedding VDCs and the substrate network. This section discusses different associated components of *GAMap* and highlights their functionalities.

A. *GAMap Components*

The components of *GAMap* are discussed subsequently.

1) VDC Requests: As multiple VDCs can be re-embedded concurrently, we use $\mathcal{G} = \{\mathcal{G}^1, \mathcal{G}^2, \cdots\}$ to capture the set of all VDCs requiring re-embedding. Each VDC $\mathcal{G}^i \in \mathcal{G}$ is formally represented as an undirected weighted graph $\mathcal{G}^i = (\mathcal{N}^i, \mathcal{L}^i)$. The group of VMs is denoted by the set $\mathcal{N}^i = \{v_{i,1}, v_{i,2}, \cdots\}$ and $|\mathcal{N}^i|$ is the aggregate VMs corresponding to VDC $\mathcal{G}^i$. A VL between VM $v_{i,j}$ and and a adjacent VM $v_{i,j'}$ is represented as $e^i_{j,j'}$ and $|\mathcal{L}^i|$ captures all such VLs for VDC $\mathcal{G}^i$. As shown in Figure 1, a VDC comprises VMs and VLs, each requiring different types and resources. For instance, a VM's resource needs CPU and memory resources, often represented as a unidimensional demand expressed as computational resource blocks (CRBs) [5]. In the context of *GAMap*, the commencing CRB demand of $v_{i,j} \in \mathcal{N}^i$ is represented as $d_{i,j}$. On the other hand, the commencing demand of a VL $e^i_{j,j'}$ is denoted as $b(e^i_{j,j'})$. As *GAMap* encompasses dynamic re-embedding of VDCs, the updated resource demands of $v_{i,j}$ are denoted by $\overline{d}_{i,j}$. In contrast, the upgraded demand of $e^i_{j,j'}$ are reflected as $\overline{b}(e^i_{j,j'})$.

2) Substrate Network: An interconnected substrate network is represented as an undirected weighted graph $\mathcal{G}^\mathcal{S} = (\mathcal{N}^\mathcal{S}, \mathcal{L}^\mathcal{S})$. The node set $\mathcal{N}^\mathcal{S}$ comprises the set of servers $\mathcal{N}^\mathcal{H}$ and the set of routers $\mathcal{N}^\mathcal{R}$. We assume that only the servers have computational resources, also expressed in CRBs. The capacity of a server $h_k \in \mathcal{N}^\mathcal{H}$ is reflected as t_k, and c_k is the current available capacity. Further it is assumed that $t_k = c_k, \forall h_k \in \mathcal{N}^\mathcal{H}$ initially. The cost of using resources at h_k is p_k. Any two servers $h_k, h_{k'}$ have unique server capacities, i.e., $t_k \neq t_{k'}$, thereby enforcing a heterogeneous setup. The substrate network also consists of multiple physical links captured as $\mathcal{L}^\mathcal{S}$. Let $c(e_l)$ and $b(e_l)$ be the total and available capacities of a physical link $e_l \in \mathcal{L}^\mathcal{S}$ having a unit bandwidth cost of p_l. Provisioning resources for $e^i_{j,j'}$ encompasses reserving $b(e^i_{j,j'})$ bandwidth resource on a simple substrate path $\delta_{h_k,h_{k'}} \in \mathbb{P}_{h_k,h_{k'}}$. Note that $\mathbb{P}_{h_k,h_{k'}}$ represents the set of simple paths between h_k and $h_{k'}$. Moreover, the bandwidth of $\delta_{h_k,h_{k'}}$ is denoted as $b(\delta_{h_k,h_{k'}})$.

B. *Working of GAMap*

The working of *GAMap* can be captured using the following modules: (*i.*) *initial population generator*, (*ii.*) *GA-Engine*, and (*iii.*) *best individual selector*. Once the VDCs to be re-embedded are identified, the *initial population generator* module is invoked. It generates a population of individuals, each representing a feasible re-embedding of the VDCs. The details of the initial population generation are provided in **Section V-B1**. The initial population then acts as an input to the *GA-Engine* that is responsible for generating high-quality re-embedding solutions using iterative selection, crossover, and mutation operators and is discussed elaborately in *Section V-B2*. Finally, the *best individual selector* outputs the least cost re-embedding plan catering to the design objective of *GAMap*.

IV. Problem Formulation

A SP can re-embed the VDCs after its initial deployment to manage resource demand spikes, balance the load, effectively utilize substrate resources, and oversee hardware failures [2]. This paper focuses on developing an efficient strategy to relocate the VDCs completely. Owing to the inherent correlation among the VMs, a complete relocation implies relocating all the VMs and the VLs corresponding to the relocating VDC. The relocation of VDCs encompasses two stages. In the initial stage, the VMs corresponding to VDCs is relocated. The final stage involves finding feasible paths satisfying the bandwidth demands of VLs between the already relocated VMs in the first stage. The overall objective of this work is to minimize the re-embedding cost. The relocation policy must adhere to some constraints that are discussed subsequently.

A. Relocation Constraints

The re-embedding must abide by the following constraints, (*i.*) VM relocation, (*ii.*) VL Relocation, (*iii.*) Completeness of re-embedding, and (*iv.*) Assignment.

1) VM Relocation Constraint: The *VM relocation constraint* states that any VM $v_{i,j} \in \mathcal{N}^i$ can be successfully re-embedded to h_k if Equation (1) is fulfilled.

$$\overline{d}_{i,j} \leq c_k \tag{1}$$

2) VL Relocation Constraint: Considering $v_{i,j}$ and $v_{i,j'}$ embedded on servers h_k and $h_{k'}$, respectively connected via $e^i_{j,j'}$ can be re-embedded to substrate path $\delta_{h_k,h_{k'}} \in \mathbb{P}_{h_k,h_{k'}}$ if Equation (2) is satisfied.

$$\overline{b}(e^i_{j,j'}) \leq b(\delta_{h_k,h_{k'}}) \tag{2}$$

Binary Indicators: An indicator variable $\mathcal{X}(v_{i,j}, h_k)$ corresponding to $v_{i,j}$ is defined as per Equation (3).

$$\mathcal{X}(v_{i,j}, h_k) = \begin{cases} 1 & \text{If } v_{i,j} \text{ is successfully re-embedded to } h_k \\ 0 & \text{Otherwise} \end{cases} \tag{3}$$

On similar grounds, let $\mathcal{X}(e^i_{j,j'}, \delta_{h_k,h_{k'}})$ be a path indicator variable for $e^i_{j,j'}$ and captured as Equation (4).

$$\mathcal{X}(e^i_{j,j'}, \delta_{h_k,h_{k'}}) = \begin{cases} 1 & \text{Path } \delta_{h_k,h_{k'}} \text{ is assigned} \\ & \text{for } e^i_{j,j'} \text{and} \quad \mathcal{X}(v_{i,j'}, h_k) = 1 \\ & \textit{and } \mathcal{X}(v_{i,j}, h_{k'}) = 1 \\ & \\ 0 & \text{Otherwise} \end{cases} \tag{4}$$

3) Assignment Constraint: Considering $v_{i,j}$ and $v_{i,j'}$ for re-embedding, they should abide by the assignment constraint as per Equation (5). The re-embedding should ensure that no two VMs of a VDC are re-embedded onto the same server. This constraint follows the models discussed in [2], [10]. The overall agenda of introducing this constraint is to assist in providing distributed services and avoid single-point failures.

$$\mathcal{X}(v_{i,j'}, h_k) \wedge \mathcal{X}(v_{i,j}, h_k) = 0 \tag{5}$$

4) Completeness of Re-embedding: A VDC is considered to be completely embedded if every constituent VM and VL is allocated resource and is reflected in Equations (6) and (7).

$$|\mathcal{N}^i| = \sum_{\forall v_{i,j} \in \mathcal{N}^i} \sum_{\forall h_k \in \mathcal{N}^{\mathcal{H}}} \mathcal{X}(v_{i,j}, h_k) \tag{6}$$

$$|\mathcal{L}^i| = \sum_{\forall e^i_{j,j'} \in \mathcal{L}^i} \sum_{\forall \delta_{h_k,h_{k'}} \in \mathbb{P}_{h_k,h_{k'}}} \mathcal{X}(e^i_{j,j'}, \delta_{h_k,h_{k'}}) \tag{7}$$

B. Re-embedding Cost Computation

GAMap aims to reduce the total re-embedding cost for relocating VDCs. For a VDC $\mathcal{G}^i$, the remapping cost comprises *VM relocation cost* and *VL relocation cost*. We subsequently present a formal representation of its computation.

1) VM Re-embedding Cost: Re-embedding cost $\psi^k_{i,j}$ of $v_{i,j}$ on h_k is captured as Equation (8).

$$\psi^k_{i,j} = \overline{d}_{i,j} * p_k \tag{8}$$

2) VL Re-embedding Cost: Re-embedding cost of $e^i_{j,j'}$ onto path $\delta_{h_k,h_{k'}}$ is derived as Equation (9).

$$\lambda^i_{j,j'}(\delta_{h_k,h_{k'}}) = \sum_{\forall e_l \in \delta_{h_k,h_{k'}}} \overline{b}(e^i_{j,j'}) * p_l \tag{9}$$

3) Total Remapping Cost: From Equations (8) and (9) the total remapping cost η_i for relocating a VDC $\mathcal{G}^i$ can be derived as follows:

$$\begin{aligned} \eta_i = & \sum_{\forall v_{i,j} \in \mathcal{N}^i} \sum_{\forall h_k \in \mathcal{N}^{\mathcal{S}}} \mathcal{X}(v_{i,j}, h_k) * \psi^k_{i,j} \\ & + \sum_{\forall e^i_{j,j'} \in \mathcal{L}^i} \sum_{\forall \delta_{h_k,h_{k'}} \in \mathbb{P}_{h_k,h_{k'}}} \mathcal{X}(e^i_{j,j'}, \delta_{h_k,h_{k'}}) * \lambda^i_{j,j'}(\delta_{h_k,h_{k'}}) \end{aligned} \tag{10}$$

C. Objective

The overall objective of *GAMap* is captured in Equation (11a). Constraint (11b) is the *VM relocation constraint*, stating that there should be at least one server in $h_k \in \mathcal{N}^{\mathcal{H}}$ satisfying the demand of $v_{i,j}$. Similarly, Constraint (11c) is the *VL relocation constraint*. It expresses that there should be at least one possible path $p_{h_k,h_{k'}}$ meeting the bandwidth requirement of $e^i_{j,j'}$. A VM is re-embedded to no more than one server, represented as Constraint (11d). Constraints (11e) and (11f) dictate the *completeness* of re-embedding for a VDC. The *assignment constraint* states that no more than one VM corresponding to a VDC should be re-embedded to a server [2] and this is formally defined as Constraint (11g). Constraints (11h) and (11i) are values of variables.

$$\text{minimize} \sum_{\forall \mathcal{G}^i \in \mathcal{G}} \eta_i \tag{11a}$$

$$\text{s.t.} \quad \overline{d}_{i,j} \leq c_k \tag{11b}$$

$$\overline{b}(e^i_{j,j'}) \leq b(\delta_{h_k,h_{k'}}) \tag{11c}$$

$$\sum_{\forall h_k \in \mathcal{N}^{\mathcal{H}}} \mathcal{X}(v_{i,j}, h_k) = 1 \tag{11d}$$

$$|\mathcal{N}^i| = \sum_{\forall v_{i,j} \in \mathcal{N}^i} \sum_{\forall h_k \in \mathcal{N}^{\mathcal{S}}} \mathcal{X}(v_{i,j}, h_k) \tag{11e}$$

$$|\mathcal{L}^i| = \sum_{\forall e^i_{j,j'} \in \mathcal{L}^i} \sum_{\forall \delta_{h_k,h_{k'}} \in \mathbb{P}_{h_k,h_{k'}}} \mathcal{X}(e^i_{j,j'}, \delta_{h_k,h_{k'}}) \tag{11f}$$

$$\mathcal{X}(v_{i,j'}, h_k) \wedge \mathcal{X}(v_{i,j}, h_k) = 0 \tag{11g}$$

$$\forall h_k \in \mathcal{N}^{\mathcal{S}}; \quad \forall e^i_{j,j'} \in \mathcal{L}^i; \quad \forall\, v_{i,j}, v_{i,j'} \in \mathcal{N}^i \tag{11h}$$

$$\forall \delta_{h_k,h_{k'}} \in \mathbb{P}_{h_k,h_{k'}}; \quad \forall\, \mathcal{G}^i \in \mathcal{G} \tag{11i}$$

D. NP-Hardness

The VDC re-embedding problem, as expressed in Equation (11), is $\mathcal{NP}$-Hard and closely resembles the multi-way separator problem [5]. Obtaining an optimal solution for VNE essentially implies solving two computationally complex problems of VM and VL re-embedding. Moreover, owing to the intractable nature of the problems, there is an obvious trade-off between *solution quality* and *computational time* with increasing problem size. In this regard, the literature on VDC management has adopted three different approaches,

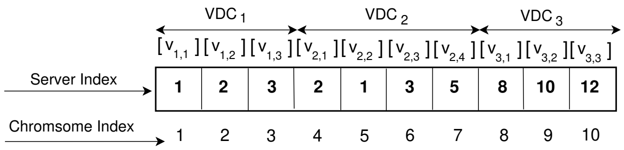


Fig. 4: Chromosome Structure.

i.e., (*i.*) *exact*, (*ii.*) *heuristics*, and (*iii.*) *meta-heuristics*. The *exact* approaches generate optimal solutions for small problem instances. They are *compute-intensive*, *time-consuming*, and *non-scalable* for larger test cases, thereby forbidding their adoption. On the other hand, they often compromise the solution quality to obtain a quick solution and tend to be stuck at the local optimum. To overcome these issues, meta-heuristic-based solutions have been proposed that consume relatively more time than the heuristic-based ones but provide sub-optimal or closer to optimal solutions to the global optimum.

V. Solution Approach

As discussed, the relocation procedure for VDCs is proven intractable. Hence, this work adopts a genetic algorithm (GA)–based strategy to generate a *sub-optimal* re-embedding plan to reduce the remapping costs. GA is a bio-inspired technique motivated by the process of natural selection. It has been adapted to solve multiple problems in cloud environments [18]–[21]. Before deep diving into the workings of GA, a brief insight into some relevant operators is provided.

A. Background of GA

GA is adapted to generate high-quality solutions for optimization and search problems by banking on bio-inspired operators such as *mutation*, *crossover*, and *selection* [12]. Each operator plays a vital role in boosting the solution quality, and its significance is discussed subsequently.

Definition 1: **(Selection Operator)**: The *selection operator* assists in identifying and eliminating inferior solutions and retaining, replicating fitter or superior solutions [12].
The retained solutions can then reproduce, improving convergence and solution quality. A fitness value assigned to each distinguishes between good and bad solutions. Different *selection operators* such as *tournament selection*, *roulette-wheel selection*, *proportionate selection*, and *rank selection*, and a detailed discussion on the same can be found in [12].

Definition 2: **(Crossover Operator)**: *Crossover* is used to generate new and diverse solutions by exchanging genetic information, i.e., genes between two parent chromosomes [12]. Although *crossover operations* are optional, it helps make the population diverse, avoid local optima, and achieve faster convergence. Once the parent chromosomes are identified, offspring can be generated using *single or multi-point*, or *uniform crossover operations*.

Definition 3: **(Mutation Operator)**: *Mutation* operations are used to tweak chromosomes' genetic information to introduce greater population diversity [12].
Mutation can be done in multiple ways including *bit-flip*, *swap*, and *scramble* operations.

Definition 4: **(Exploration and Exploitation)**: Exploration involves probing new search spaces to obtain diverse solutions whereas *exploitation* refers to probing a limited region of the solution at hand, thereby intensifying the local search [12].

An essential aspect of generating high-quality solutions using GA is the balance between *exploration* and *exploitation*. Interestingly, exploration in *GAMap* is attained in two ways, i.e., initial population and exploration via genetic operators, i.e., *crossover* and *mutation*. Exploration in the initial population focuses on diversifying at the beginning of GA, and exploration in *crossover/mutation* occurs during the evolution process and involves creating new solutions from existing ones. Both aspects contribute to the overall exploration of the search space in search for optimality. Note that the selection operator helps achieve exploitation of the search space.

B. GA-Based Relocation Strategy

This section provides a detailed insight into the proposed scheme. Firstly, the encoding and initial population generation strategies are presented. This is followed by a detailed discussion on *crossover* and *mutation* operations. Finally, the GA-based relocation procedure is discussed.

1) Encoding and Initial Population Generation: Two of the most critical aspects of any GA-based solution strategy are the encoding of individuals and initial population generation.
Encoding: A chromosome depicting a successful relocation of VDCs is shown in Figure 4. The figure captures a feasible relocation of three VDCs and their corresponding VMs. In this context, a feasible allocation implies that the VMs and the corresponding VLs of the VDCs are relocated successfully. It is to be noted that a natural encoding scheme is adopted to capture a feasible allocation [10]. Referring to Figure 4, the value 1 at gene index 1 of the chromosome represents the server onto which the VM $v_{1,1}$ is remapped. The VMs of the VDCs are sequentially represented in the chromosome to reduce the implementation complexity. It can also be inferred from Figure 4 that the genes corresponding to the VDCs follow the indexing of the VDCs. Moreover, the chromosome size corresponds to the VMs considering all relocating VDCs.
Initial Population Generation: The initial population comprises 50% randomly generated solutions, whereas the remaining 50% are copies of a greedy-VM allocation policy followed by shortest path VL embedding. The goal of dividing the initial population is to balance the *exploration* and *exploitation* of the solution space. All randomly generated individuals in the population will have broader *exploration* but will suffer from delayed convergence and poor-quality solutions. On the other hand, *exploitation* is achieved using heuristics, but it may not ensure a global optimum. Therefore, the partition mentioned above is enforced to achieve a trade-off. Note that the random chromosomes also represent a feasible assignment of VDCs.
2) Crossover and Mutation Operation: The crossover and mutation operations are detailed next.
Crossover: The crossover operations in the literature are limited to single or multi-point. However, these crossover operators are more adapted to binary-coded populations than naturally encoded populations [12]. Though the overall working

This article has been accepted for publication in IEEE Transactions on Green Communications and Networking. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TGCN.2023.3345542

Algorithm 1: GA Based VDC Re-embedding.

Input: $\mathcal{G}$, $\mathcal{G}^{\mathcal{S}}$.
Result: C_f, The best re-embedding solution, i.e., chromosome.
Initialize: $P = \Phi$
$P \leftarrow$ **InitialPopulation**($\mathcal{G}$, $\mathcal{G}^{\mathcal{S}}, P_{size}$)
for $i = 1 \ to \ N_{gen}$ **do**
  for $j = 1 \ to \ P_{size}$ **do**
    Select two chromosomes C_1 and C_2 using tournament selection
    $P' \leftarrow$ **ImprovedCrossover**(C_1, C_2)
    for *each* $C_a \in P'$ **do**
      if ***!CheckFeasiblity***(C_a) **then**
        $P' = P' \setminus \{C_a\}$
    for *each* $C_a \in P'$ **do**
      r' = Rand[0, 1]
      if $r < r'$ **then**
        C'_a = **Mutation**(C_a)
        if ***CheckFeasiblity***(C'_a) **then**
          $P = P \cup \{C'_a\}$
      else
        $P = P \cup \{C_a\}$
  $P \leftarrow$ **ImportElites**(P)
$C_f \leftarrow$ **SelectBest**(P)

Algorithm 2: Initial Population.

Input: $\mathcal{G}$, $\mathcal{G}^{\mathcal{S}}$, and P_{size}
Result: $P_{initial}$, The initial population.
Initialize: $P_{initial} = \Phi$, $P_{count} = 0$
while $P_{count} < P_{size}$ **do**
  if $P_{count} < P_{size}/2$ **then**
    if ***RandomSolution*** *($\mathcal{G}$, $\mathcal{G}^{\mathcal{S}}$)* **then**
      $P_{initial}$ = $P_{initial}$ $\cup$ **GetChromsome**()
      P_{count}++
  else
    if *Greedy ($\mathcal{G}$, $\mathcal{G}^{\mathcal{S}}$)* **then**
      $P_{initial}$ = $P_{initial}$ $\cup$ **GetChromsome**()
      P_{count}++
return $P_{initial}$

Algorithm 3: Random Solution

Input: $\mathcal{G}$ and $\mathcal{G}^{\mathcal{S}}$
Result: Boolean
1 **Initialize**: $free[v_{i,j}] = T$, $\forall\, v_{i,j} \in \mathcal{N}^i$, $\forall \mathcal{G}^i \in \mathcal{G}$
2 **for** *every* $\mathcal{G}^i \in \mathcal{G}$ **do**
3   **for** *every* $v_{i,j} \in \mathcal{N}^i$ **do**
4   count = 0
5   **while** $free[v_{i,j}] = T$ *and count* $\leq$ *threshold* **do**
6   k = Rand [0, $|\mathcal{N}^{\mathcal{H}}|$]
7   count++
8   **if** $\overline{d}_{i,j} \leq c_k$ *and* $\nexists\ v_{i,j'} \mid \mathcal{X}(v_{i,j'}, h_k)$ = 1 **then**
9   c_k = c_k - $\overline{d}_{i,j}$
10        $free[v_{i,j}]$ = F
11   **SetChromosome** $(v_{i,j}, h_k)$
12   **if** $free[v_{i,j}] == T$ **then**
13   return F
14 **for** *every* $\mathcal{G}^i \in \mathcal{G}$ **do**
15   **for** *every* $e^i_{j,j'} \in \mathcal{L}^i$ **do**
16   **if** ***!FeasiblePath*** $(\mu(v_{i,j}), \mu(v_{i,j'}))$ **then**
17   return F
18 return T

of the crossover in *GAMap* is motivated by the work in [10], it has multiple points of differences, as previously mentioned in Section II. Motivated from [10], this work adopts *improved crossover* to generate distinct offspring, favoring the *exploration* of the solution space. A *tournament selection* procedure is adopted to select the parent chromosomes for *crossover*, ensuring that fitter individuals participate in reproduction to generate better-quality offspring. Two-parent chromosomes C_1 and C_2 are selected via *tournament selection*. Next, the similarity index $SIM(C_1, C_2)$ between them is computed. The similarity $SIM(C_1, C_2)$ is also a chromosome wherein each gene entry is set to 1 if the genetic information is the same for both parents, 0 otherwise. For instance, in Figure 4, an entry 3 at the chromosome index 3 for both parents implies that the 3^{rd} index in $SIM(C_1, C_2)$ is set to 1, implying the same genetic information. Next, the hamming distance $H(C_1, C_2)$ indicating the number of 1's in $SIM(C_1, C_2)$ is calculated. The *improved crossover* operator then appropriately selects the *crossover* point depending on $H(C_1, C_2)$. In this context, if $H(C_1, C_2) > 2$, then a *single point crossover* operation is performed by selecting a point anywhere on the scale $\alpha(C_1, C_2)$. Here, $\alpha(C_1, C_2)$ is a range starting with the occurrence of the first 1 to the last 1, respectively. On the other hand, if $H(C_1, C_2) \leq 2$, the *crossover* operation is not performed, and selected parents directly proceed for mutation. The findings in [22] show that *crossover* operations using hamming distances generate distinct offspring.

Mutation: The *improved crossover* operation is followed by a *bit-flop mutation* adding diversity and favoring exploration [12]. It proceeds by generating a random gene corresponding to its server allocation, is selected, and is replaced by a random server index satisfying the VM's and adjacent VL's demand.

Note that the *crossover* and *mutation* operations are performed on the population across generations, which sometimes can result in infeasible solutions, resulting in delayed convergence. However, the convergence delay incurred due to genetic operators is less severe than the initial population.

C. GA Based Remapping Procedure

The overall working of the GA-based remapping procedure is shown in **Algorithm 1**. The algorithm takes as input the relocating VDCs $\mathcal{G}$ and substrate network $\mathcal{G}^{\mathcal{S}}$. The algorithm outputs a feasible re-embedding of the VDCs as a chromosome C_f. Algorithm 1 is initiated with the initial population generation as shown in **Algorithm 2**. It returns P_{size} feasible re-embedding solutions comprising $P_{size}/2$ randomly generated solutions, and the remaining $P_{size}/2$ are copies of a minimum cost-based greedy VM allocation followed by the shortest path re-embedding for VLs. The random allocation procedure is captured as **Algorithm 3**. It proceeds as follows. Initially, all the VMs $v_{i,j} \in \mathcal{N}^i$ corresponding to a VDC $\mathcal{G}^i \in \mathcal{G}$ are free, i.e., $free[v_{i,j}] = T$, indicating its availability in the relocation process (Step 1 of **Algorithm 3**). The algorithm then operates in two phases, (*i.*) VM allocation, followed by (*ii.*) VL allocation. In the first phase, every free VM $v_{i,j}$ of a VDC $\mathcal{G}^i$ generates a random server index k in the pre-defined

Algorithm 4: Greedy

Input: $\mathcal{G}$ and $\mathcal{G}^{\mathcal{S}}$
Result: Boolean
Initialize: $free[v_{i,j}] = F$, $\forall\, v_{i,j} \in \mathcal{N}^i$, $\forall \mathcal{G}^i \in \mathcal{G}$
Sort all the servers $h_k \in \mathcal{N}^{\mathcal{H}}$ in increasing order of p_k and store it in $\mathbb{SL}$
for *every* $\mathcal{G}^i \in \mathcal{G}$ **do**
    for *every* $v_{i,j} \in \mathcal{N}^i$ **do**
        while $free[v_{i,j}] = T \;\; | \;\; \mathbb{SL}! = \Phi$ **do**
            Let h_k be the first unprocessed server in $\mathbb{SL}$
            if $\overline{d}_{i,j} \leq c_k$ *and* $\nexists\, v_{i,j'} \mid \mathcal{X}(v_{i,j'}, h_k) = 1$ **then**
                $c_k = c_k - \overline{d}_{i,j}$
                free$[v_{i,j}] = F$
                SetChromosome $(v_{i,j}, h_k)$
        if $free[v_{i,j}] == T$ **then**
            return F
for *every* $\mathcal{G}^i \in \mathcal{G}$ **do**
    for *every* $e^i_{j,j'} \in \mathcal{L}^i$ **do**
        if *!**FeasiblePath*** $(\mu(v_{i,j}), \mu(v_{i,j'}))$ **then**
            return F
return T

range $[0, |\mathcal{N}^{\mathcal{H}}|]$. As the randomly generated server h_k may not be feasible, a *count* variable is maintained to keep track of the threshold for such attempts. Depending on the state of h_k, either of the following scenarios arises: (*i.*) allocation is made or (*ii.*) allocation is not made. In the former case, $v_{i,j}$ is assigned to h_k if: (*i.*) h_k has enough resources to host $v_{i,j}$ and (*ii.*) $\nexists\, v_{i,j'}$ such that $\mathcal{X}(v_{i,j'}, h_k)$ = 1. Further, the mapping is also added to the chromosome using the **SetChromsome**(.) method (Steps 8-11 of **Algorithm 3**).

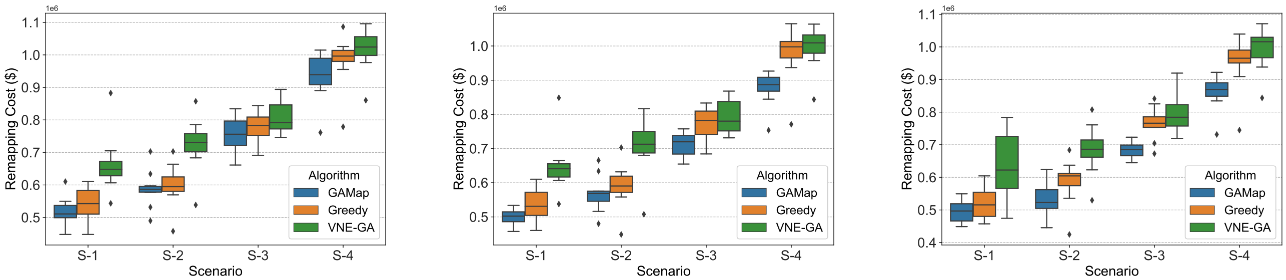
In the latter case, as $v_{i,j}$ is free, i.e., $free[v_{i,j}] = T$, a new attempt to allocate $v_{i,j}$ following the procedure discussed-above is repeated. Moreover, suppose $v_{i,j}$ remains unassigned after a maximum number of allowed attempts. In that case, the VM allocation procedure is abruptly terminated, and a F value, indicating a failed attempt, is returned (Steps $12-13$ of **Algorithm 3**). The failure results in the initiation of a renewed attempt at assigning substrate resources to all the VMs. On completing the VM allocation phase, the VL allocation phase is initiated wherein each VL $e^i_{j,j'} \in \mathcal{L}^i$ is mapped onto a feasible substrate path. The **FeasiblePath** (.) procedure returns a T value in case of successful allocation, between $\mu(v_{i,j})$ and $\mu(v_{i,j'})$, where $\mu(.)$ returns the allocated server of the corresponding VM. The procedure is prematurely terminated if the VL fails and VM allocation is started afresh.

Once $P_{size}/2$ random feasible solutions are generated, the greedy resource allocation policy as shown in **Algorithm 4** is triggered. The only point of difference from the random allocation strategy is that the substrate servers are sorted in non-decreasing order of their hosting costs p_k and stored in $\mathbb{SL}$. The VM allocation policy also processes the hosts in the order $\mathbb{SL}$. The rest is executed as per random allocation policy. The feasible solutions/chromosomes obtained via random/greedy allocation are added to the initial population using the **GetChromsome** (.) procedure. Note that the procedure compiles the feasible solutions and converts them into a chromosome representation, as reflected in Figure 4, reflecting its assignment. Once converted, these solutions are added to the initial population, which is genetically modified to generate new assignments. Once p_{size} solutions are generated, **Algorithm 2** is terminated, and the initial population denoted by $P_{initial}$ is returned. With the initial population as input, the GA-based VDC Remapping procedure commences. It iterates over N_{gen} rounds and outputs a feasible re-embedding with minimum remapping cost. Subsequently, improved crossover and mutation operations are performed in every round to identify new individuals. The process of generating new individuals starts with a *tournament selection*, wherein two of the fittest chromosomes, say C_1 and C_2, in $P_{initial}$ are identified for the **ImprovedCrossover** (.) operation (Step 5 of **Algorithm 1**). The details of the improved crossover operation can be found in Section V-B2. Note that the fitness of a chromosome is computed as per Equation (11a), and a lesser aggregate cost indicates higher fitness. The *improved crossover* operator then generates new offspring that are captured as P' (Step 6 of **Algorithm 1**). Feasibility checks follow this for each chromosome $C_a \in P'$. The **CheckFeasiblity**(.) procedure verifies if VMs and VLs are mapped onto substrate servers and paths satisfying the constraints expressed in Equation (11). The infeasible solutions are ignored, whereas the feasible ones are retained in P' (Steps $7-9$ of **Algorithm 1**). Each chromosome $C_a \in P'$ is mutated next depending on the value of a random number r. This results in the following two scenarios. The first scenario occurs when $r < r'$ and the mutation operation is performed. If feasible, the mutated solution is added to P; otherwise, it is ignored. The mutation is not performed in the second case, and C_a is directly added to P (Steps $10-17$ of **Algorithm 1**). This concludes one round of the remapping procedure. This process (Steps 4-17 of **Algorithm 1**) is repeated P_{size} times, and in each iteration, individuals are added to P. Before proceeding to the next round, the fittest P_{size} individuals are identified based on the fitness function defined in Equation (11a). These P_{size} individuals are identified using the **ImportElites**(.) procedure, which sorts the chromosomes in ascending order of fitness, i.e., the remapping costs and returns the first P_{size} individuals (Step 18 of **Algorithm 1**). Finally, after executing N_{gen} rounds, the chromosome with the best fitness is identified using the **SelectBest**(.) procedure and is denoted by C_f.

D. Asymptotic Analysis of GAMap

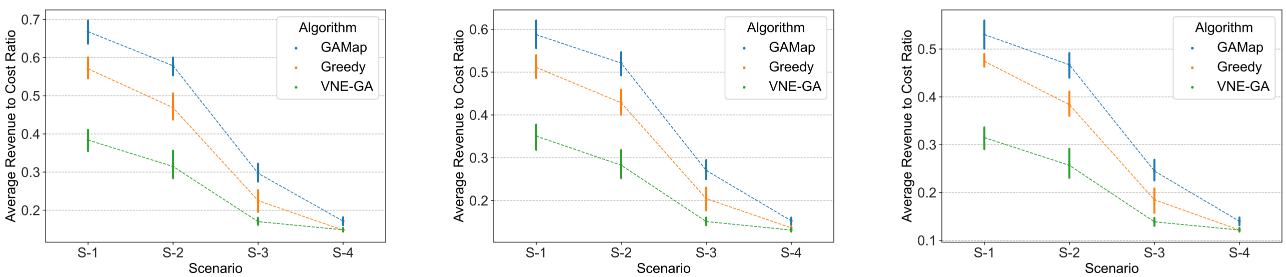
Estimating the asymptotic complexity of GA-based solutions is *extremely challenging* and is still an *open problem*. However, multiple efforts have been made to approximate roughly the running time of such an algorithm for specific cases, and it cannot be generalized for all cases. Specific to network virtualization, the work in [23] discussed the asymptotic complexities of GA. However, the closest approximation was presented in [24], and it provides the foundation for deriving the complexity of *GAMap*. The crux of *GAMap* is Algorithm 1, and the operations that consume maximum time are (*i.*) selection, (*ii.*) crossover, (*iii.*) mutation, and (*iv.*) feasibility check. Specific to a generation, the selection operation in *GAMap* uses tournament selection that consumes $O((P_{size})^2)$

This article has been accepted for publication in IEEE Transactions on Green Communications and Networking. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TGCN.2023.3345542



(a) $N_{gen} = 4$. (b) $N_{gen} = 6$. (c) $N_{gen} = 8$.

Fig. 5: Remapping Cost ($) vs. Scenarios.



(a) $N_{gen} = 4$. (b) $N_{gen} = 6$. (c) $N_{gen} = 8$.

Fig. 6: Average Revenue to Cost Ratio vs. Scenarios.

time, where P_{size} captures the population size. The crossover and mutation operations have an asymptotic complexity of $O(P_{size} * |\bigcup_{\forall \mathcal{G}^i \in \mathcal{G}} \mathcal{N}^i|)$ and $O(|\bigcup_{\forall \mathcal{G}^i \in \mathcal{G}} \mathcal{N}^i| * |\mathcal{N}^{\mathcal{S}}|)$, where $\mathcal{N}^i$ and $\mathcal{N}^{\mathcal{S}}$, respectively capture the number of VMs in the VDC and substrate servers. Finally, the feasibility involves verifying if the VLs are mapped onto substrate paths with requisite resources, which consumes $O((|\bigcup_{\forall \mathcal{G}^i \in \mathcal{G}} \mathcal{L}^i|) * |\mathcal{L}^{\mathcal{S}}|^{|\mathcal{L}^{\mathcal{S}}|})$, where $\mathcal{L}^i$ and $\mathcal{L}^{\mathcal{S}}$ respectively capture the VLs and physical links.

VI. Performance Evaluation

We use the *CloudSim* simulator toolkit [25] for experimentation, and the details are as follows.

A. Environmental Setup

For simulation, 4 interconnected DCs have been considered that are implemented as a fat-tree created using the topology generator *BRITE* [26]. The links interconnecting and top-of-rack (ToR) switches and servers are 10 *Gbps* capacity. Similarly, the physical links between the switches and DCs are assumed to be 40 *Gbps* capacity. The parameters for simulation follow the setups in [2], [10]. Additional GA-based simulation parameters are set as follows (*i.*) N_{gen} is aggregate number of iterations varies in the range $[4, 6, 8]$, (*ii.*) P_{size} is the total size of the population and is set to 8, and (*iii.*) μ captures the rate of mutation. Note that the parameters such as N_{gen} and P_{size} in GA are not pre-determined and depend on various factors, such as the nature of the optimization, characteristics of the search space, and convergence behavior. We want to state that there is no universal "one-size-fits-all" choice for the number of generations (N_{gen}). Instead, it should be tuned based on the problem characteristics and results obtained (reduced re-embedding cost in *GAMap*). Therefore, with careful analysis, trial and error is employed to find an appropriate sub-optimal solution. To assess the performance of *GAMap*, we compare it with two different baseline algorithms. Scenarios S-1, S-2, S-3, S-4 corresponding to 250, 500, 750, and 1000 VDCs respectively are considered for evaluation. Choosing such many VDCs aims to analyze how *GAMap* would potentially execute in production data centers hosting many applications. Additionally, works in [5], [13] have adopted such a test strategy to validate their respective performance. 10 test runs are conducted for each scenario, and results are compared.

B. The Baselines

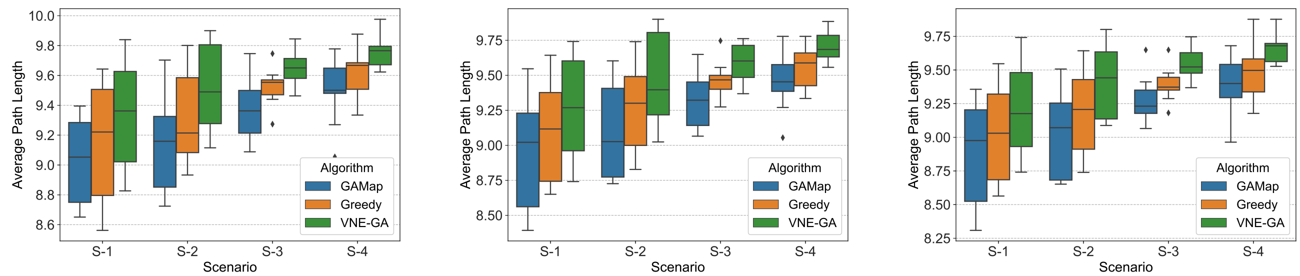
The performance of *GAMap* is compared with two baselines.

- Virtual Network Embedding based on Genetic Algorithm (VNE-GA) [14]: It adopts a GA-based embedding strategy for the virtual network (VN) over a single domain network. To generate a sub-optimal embedding, multi-point crossover and bit-flop mutation have been adopted, focused on minimal substrate resource usage.
- Greedy: In the greedy strategy, all the VMs of the VDCs are greedily assigned to the least cost available servers satisfying their resource demands, followed by the shortest path embedding of the VLs.

C. Simulation Results

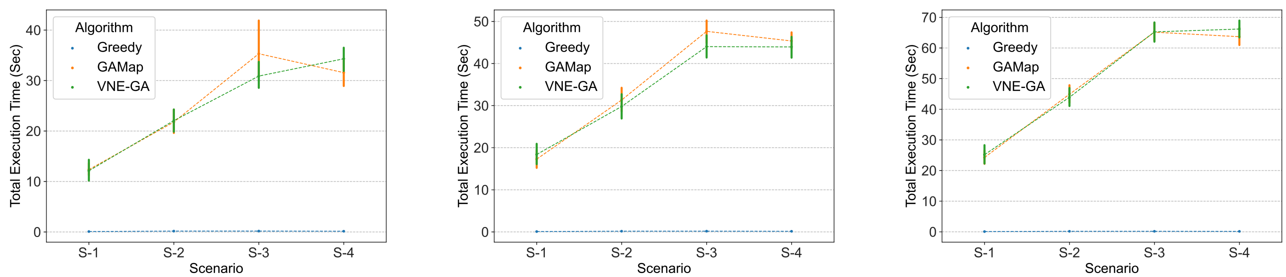
This subsection provides insight into the results obtained and their comparative behavior.

This article has been accepted for publication in IEEE Transactions on Green Communications and Networking. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TGCN.2023.3345542



(a) $N_{gen} = 4$. (b) $N_{gen} = 6$. (c) $N_{gen} = 8$.

Fig. 7: Average Path Length vs. Scenarios.



(a) $N_{gen} = 4$. (b) $N_{gen} = 6$. (c) $N_{gen} = 8$.

Fig. 8: Execution Time (s) vs. Scenarios.

TABLE I: Comparison with Optimal Solution.

	Remapping Cost ($)	
# Applications	**5 VDCs**	**10 VDCs**
Optimal	4926.9	9853.8
GAMap	6197.5	12395.1
Greedy	6525.3	13050.6
VNE-GA	8627.1	17254.1

1) Remapping cost: It indicates the total cost incurred in remapping all the relocating VMs. Figures 5(a)-5(c) capture the comparative behavior of total remapping cost for varying numbers of generations, i.e., N_{gen}. It can be observed from the figures that for less number of generations, i.e., $N_{gen} = 4$, the performance of *GAMap* and Greedy are comparable. This is because of limited exploitation due to the restricted number of generations. However, we can observe from Figures 5(b) and 5(c) that for an increasing number of generations, the performance of *GAMap* improves due to more exploitation, thereby eluding the local optima. On the other hand, VNE-GA exhibits poor performance compared to others. Individually, the total remapping cost increases with increasing VDCs, which is anticipated behavior for a small test case, i.e., 250 VDCs, *GAMap* and *Greedy* achieve similar remapping costs. However, the former outperforms the latter for larger test cases, i.e., $500 - 1000$. The poor performance of *Greedy* is attributed to the fact that it only considers VM hosting costs while assigning servers to VMs. Such an allocation strategy causes a dispersed placement of correlated VMs as the less charging servers may not always be placed in close vicinity, thereby elevating the VL embedding costs. This behavior dictates the poor performance of *Greedy* for larger test cases. On the other hand, the inferior performance of VNE-GA is attributed to the following two reasons. Firstly, the initial population is generated randomly, which does not augur well for developing reasonable quality solutions for reducing remapping costs. Secondly, VNE-GA aims to reduce the amount of substrate resources consumed, which may not necessarily result in reduced remapping costs. Although both factors contribute to its deleterious performance, the former is the primary contributor. The results in Table I depict the remapping cost for 5 and 10 VDCs. It can be observed that *GAMap* achieves lower remapping costs compared to the baselines. Additionally, we also highlight the optimal remapping cost that is computed using Google Optimization Tools [27]. We do not consider test cases with > 10 VDCs owing to the excess overheads in terms of computational time.

2) Average revenue to cost ratio: The revenue-to-cost ratio highlights the correlation between the substrate resources demanded and the number of resources provisioned for VDCs. Figures 6(a)-6(c) capture the average revenue-to-cost ratio of different techniques used for comparison. The revenue-to-cost ratio decreases for an increasing number of generations for *GAMap* and VNE-GA owing to increased opportunities for exploitation, resulting in better remapping with lower costs. However, Greedy's behavior remains consistent irrespective of generation. From Figures 6(a)-6(c), it can be inferred that an increasing number of VDCs decreases the average revenue-to-cost ratio indicating finite resources. *GAMap* achieves the highest ratio, implying that the VLs are re-embedded to relatively shorter paths. The performance of *Greedy* is inferior to *GAMap* but is superior to VNE-GA. This is primarily attributed to the scattered re-embedding of VMs in *Greedy* leading to longer physical paths for VLs. Although VNE-GA targets the reduction of resources consumed in embedding, the randomly generated initial solution space hinders achieving a good mapping. An interesting behavior is observed in scenario S-4, where all the schemes used for comparison have similar revenue-to-cost ratios. This is attributed to limited substrate resource availability in provisioning significant VDCs, thereby compelling the VLs to be mapped to inevitably longer paths.

3) Average Path Length: It reflects the average number of physical links used to provision a VL. Figures 7(a) and 7(c) illustrate the average path lengths of different protocols used for comparison across test scenarios. It can be observed from the figures that the average path lengths increase for an increasing number of VDCs, which is expected behavior. This can be attributed to the exhaustion of resources at the substrate servers leading to a dispersed placement and the exhaustion of bandwidth resources at the lower-cost shorter paths. Further, it can be observed that *GAMap* can achieve the minimum average path length compared to the baseline algorithms across all test scenarios. Further, it can also be observed that with an increasing number of generations, the average path length decreases for both *GAMap* and VNR-GA owing to the exploration of the search space. However, *GAMap* continues to outperform VNE-GA as the former can generate better quality solutions owing to the improved crossover operation instead of the classical multi-point crossover operation in the latter. Moreover, the average path length of Greedy remains relatively consistent across generations. Additionally, one of the key contributors to the reduction in the re-embedding cost in *GAMap* as depicted in Figures 5(a)-5(c) is its ability to map the VLs to comparatively shorter paths.

4) Execution Time: Figure 8(a)-8(c) illustrates the aggregate time expended by different techniques in generating a re-embedding considering different test scenarios. It is evident from the abovementioned Figures that the execution time for *GAMap* and VNE-GA continues to increase for many generations, which is anticipated behavior. Both *GAMap* and VNE-GA execute multiple times on a significant-sized population and perform time-consuming operations such as selection, crossover, and mutation, thereby incurring more runtime. It can also be observed that the greedy allocation policy reasonably quick solutions concerning *GAMap* and VNE-GA and shows consistent behavior irrespective of the number of generations.

VII. Conclusion & Future Directions

This paper proposed a model *GAMap* that constructs an efficient re-embedding of assigned VDCs experiencing resource expansion. *GAMap* develops a SP-centric solution strategy to reduce the remapping costs. To achieve efficient remapping, we adopt a GA-based solution that utilizes an *improved crossover* and *mutation* operation to generate high-quality solutions in a diverse solution space. Although *GAMap* is computer-intensive, experimental results confirm that *GAMap* can reduce the remapping cost compared to the baselines.

Although *GAMap* shows promising performance, it has scope for further enhancements. As an immediate future direction for this work, we would like to model the overall problem as a multi-objective optimization and consider additional metrics such as resource utilization and migration overheads. In addition to the re-embedding cost, incorporating topological parameters, including throughput, stress level, degrees centrality, betweenness centrality, etc., can further assist in reducing the remapping costs and balancing the load. The re-embedding procedure inevitably requires sophisticated migration strategies to schedule the migration of VMs. We would also like to integrate a migration strategy that can reduce overheads regarding migration time and downtime while generating an efficient re-embedding plan. Note that *GAMap* can seamlessly be extended for directed graphs with minor modifications.

References

[1] B. C. Camilo, R. S. Couto, and L. H. M. Costa, "Assessing the impacts of IPsec cryptographic algorithms on a virtual network embedding problem," *Computers & Electrical Engineering*, vol. 71, pp. 752–767, 2018.

[2] G. Sun, D. Liao, D. Zhao, Z. Xu, and H. Yu, "Live Migration for Multiple Correlated Virtual Machines in Cloud-Based Data Centers," *IEEE Transactions on Services Computing*, vol. 11, no. 2, pp. 279–291, 2018.

[3] K. Metwally, A. Jarray, and A. Karmouch, "A Distributed Auction-based Framework for Scalable IaaS Provisioning in Geo-Data Centers," *IEEE Transactions on Cloud Computing*, vol. 8, no. 3, pp. 647–659, 2020.

[4] Z. Li, Z. Lu, S. Deng, and X. Gao, "A Self-Adaptive Virtual Network Embedding Algorithm Based on Software-Defined Networks," *IEEE Transactions on Network and Service Management*, vol. 16, no. 1, pp. 362–373, 2019.

[5] A. Satpathy, M. N. Sahoo, L. Behera, and C. Swain, "ReMatch: An Efficient Virtual Data Center Re-Matching Strategy Based on Matching Theory," *IEEE Transactions on Services Computing*, vol. 16, no. 2, pp. 1373–1386, 2023.

[6] C. Guo, K. Xu, G. Shen, and M. Zukerman, "Temperature-Aware Virtual Data Center Embedding to Avoid Hot Spots in Data Centers," *IEEE Transactions on Green Communications and Networking*, vol. 5, no. 1, pp. 497–511, 2021.

[7] T. M. Nam, N. V. Huynh, L. Q. Dai, and N. H. Thanh, "An Energy-Aware Embedding Algorithm for Virtual Data Centers," in *2016 28th International Teletraffic Congress (ITC)*, 2016, pp. 18–25.

[8] T. M. Nam, N. H. Thanh, H. T. Hieu, N. T. Manh, N. V. Huynh, and H. D. Tuan, "Joint network embedding and server consolidation for energy–efficient dynamic data center virtualization," *Computer Networks*, vol. 125, pp. 76–89, 2017.

[9] M. F. Zhani, Q. Zhang, G. Simona, and R. Boutaba, "VDC Planner: Dynamic migration-aware Virtual Data Center embedding for clouds," in *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, 2013, pp. 18–25.

[10] B. Dab, I. Fajjari, N. Aitsaadi, and G. Pujolle, "VNR-GA: Elastic virtual network reconfiguration algorithm based on genetic metaheuristic," in *2013 IEEE Global Communications Conference (GLOBECOM)*, 2013, pp. 2300–2306.

[11] M. F. Bari, M. F. Zhani, Q. Zhang, R. Ahmed, and R. Boutaba, "CQNCR: Optimal VM migration planning in cloud data centers," in *2014 IFIP Networking Conference*, 2014, pp. 1–9.

[12] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," 1988.

[13] A. Satpathy, M. N. Sahoo, A. K. Sangaiah, C. Swain, and S. Bakshi, "CoMap: An efficient virtual network re-mapping strategy based on coalitional matching theory," *Computer Networks*, vol. 216, p. 109248, 2022.

[14] I. Pathak and D. P. Vidyarthi, "A model for virtual network embedding across multiple infrastructure providers using genetic algorithm," *Science China Information Sciences*, vol. 60, no. 4, pp. 1–12, 2017.

[15] A. Satpathy, M. N. Sahoo, A. Mishra, B. Majhi, J. J. Rodrigues, and S. Bakshi, "A Service Sustainable Live Migration Strategy for Multiple Virtual Machines in Cloud Data Centers," *Big Data Research*, vol. 25, 2021.

[16] G. Sun, D. Liao, V. Anand, D. Zhao, and H. Yu, "A new technique for efficient live migration of multiple virtual machines," *Future Generation Computer Systems*, vol. 55, pp. 74–86, 2016.

This article has been accepted for publication in IEEE Transactions on Green Communications and Networking. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TGCN.2023.3345542

[17] F. Callegati and W. Cerroni, "Live Migration of Virtualized Edge Networks: Analytical Modeling and Performance Evaluation," in *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*, 2013, pp. 1–6.

[18] F.-H. Tseng, X. Wang, L.-D. Chou, H.-C. Chao, and V. C. M. Leung, "Dynamic Resource Prediction and Allocation for Cloud Data Center Using the Multiobjective Genetic Algorithm," *IEEE Systems Journal*, vol. 12, no. 2, pp. 1688–1699, 2018.

[19] F. Hoseiny, S. Azizi, M. Shojafar, F. Ahmadiazar, and R. Tafazolli, "PGA: A Priority-aware Genetic Algorithm for Task Scheduling in Heterogeneous Fog-Cloud Computing," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2021, pp. 1–6.

[20] I. M. Ali, K. M. Sallam, N. Moustafa, R. Chakraborty, M. Ryan, and K.-K. R. Choo, "An automated task scheduling model using non-dominated sorting genetic algorithm II for fog-cloud systems," *IEEE Transactions on Cloud Computing*, vol. 10, no. 4, pp. 2294–2308, 2020.

[21] X. Gao, R. Liu, and A. Kaushik, "Hierarchical Multi-Agent Optimization for Resource Allocation in Cloud Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, pp. 692–707, 2021.

[22] Q.-y. Zhang and S.-c. Chang, "An Improved Crossover Operator of Genetic Algorithm," in *2009 Second International Symposium on Computational Intelligence and Design*, vol. 2, 2009, pp. 82–86.

[23] P. Martinez-Julia, V. P. Kafle, and H. Asaeda, "Achieving Optimized Embedding of Virtual Networks by Incremental Evolution," in *2022 25th Conference on Innovation in Clouds, Internet and Networks (ICIN)*. IEEE, 2022, pp. 121–125.

[24] P. Zhang, H. Yao, M. Li, and Y. Liu, "Virtual network embedding based on modified genetic algorithm," *Peer-to-Peer Networking and Applications*, vol. 12, pp. 481–492, 2019.

[25] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

[26] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: An approach to universal topology generation," in *MASCOTS 2001, Proceedings Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2001, pp. 346–353.

[27] OR-Tools - Google Optimization Tools. Last Accessed: December 2023. [Online]. Available: https://github.com/google/or-tools

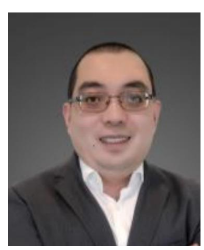
Anurag Satpathy is a Post-Doctoral Fellow in the Department of Computer Science at Missouri University of Science and Technology, Rolla, MO, USA. Before this, he completed his Ph.D. in computer science and Engineering from the National Institute of Technology, Rourkela, India. His research interests include Cloud Computing, the Internet of Things, and Distributed Systems.

Manmath Narayan Sahoo He received his Ph.D. in Computer Science in 2014 from NIT Rourkela, India. He is an Assistant Professor in the Computer Science and Engineering Department at NIT Rourkela, India. He has been a reviewer, guest editor, track chair, and program chair in many journals and conferences. His research interests include mobile ad hoc networks, fault tolerance, sensor networks, and cloud computing.

Chittaranjan Swain completed his M.Tech. degree in CSE from VIT University, Vellore, India. He is currently a research scholar in the Department of Computer Science and Engineering at the NIT Rourkela, India. His research interests include Fog Computing, Cloud Computing, and IoT.

Muhammad Bilal (Senior Member, IEEE) received the Ph.D. degree in information and communication network engineering from the School of Electronics and Telecommunications Research Institute (ETRI), Korea University of Science and Technology, in 2017. From 2017 to 2018, he was with Korea University, where he was a Postdoctoral Research Fellow with the Smart Quantum Communication Center. In 2018, he joined Hankuk University of Foreign Studies, South Korea, where he was an Associate Professor with the Division of Computer and Electronic Systems Engineering. In 2023, Dr. Bilal became part of Lancaster University, where he now serves as a Senior Lecturer (Associate Professor) within the School of Computing and Communications. Throughout his career, Dr. Bilal has been actively involved in various research projects. Dr. Bilal is a prolific author, known for his wide-ranging contributions to numerous articles published in internationally renowned, top-tier journals. His pioneering work has also led to the successful acquisition of multiple US and Korean patents. His research interests encompass Network Optimization, Cyber Security, the Internet of Things, Vehicular Networks, Information-Centric Networking, Digital Twin, Artificial Intelligence, and Cloud/Fog Computing. Dr. Bilal is an esteemed member of the editorial boards for several prominent journals, including IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, IEEE INTERNET OF THINGS JOURNAL, IEEE Future Directions in Technology, Policy, and Ethics Newsletter, Alexandria Engineering Journal (Elsevier), and Physical Communication (Elsevier). He also serves as the Co-Editor-in-Chief of the International Journal of Smart Vehicles and Smart Transportation. In addition, he has actively contributed as a Technical Program Committee Member for leading international conferences such as the IEEE Vehicular Technology Conference (VTC), the IEEE International Conference on Communications (ICC), ACM SigCom, and the IEEE Consumer Communications & Networking Conference (CCNC).

Sambit Bakshi (Senior Member, IEEE) Sambit Bakshi is currently with the Department of Computer Science and Engineering, National Institute of Technology Rourkela, India. He also serves as Head of the Centre for Automation Technology at National Institute of Technology Rourkela. His area of interest includes surveillance, biometric security, digital forensics, and social security analytics. He presently serves as associate editor of IEEE Transactions on Automation Science and Engineering, IEEE Transactions on Computational Social Systems, IEEE Transactions on Technology and Society, IEEE Transactions on Education and IEEE Systems Journal. He previously served as Associate Editor of IEEE IT Professional magazine, IEEE Access, Innovations in Systems and Software Engineering Springer: A NASA Journal, Expert Systems with Applications, Engineering Applications of Artificial Intelligence, Image and Vision Computing, Multimedia Systems, Multimedia Tools and Applications, and Neural Processing Letters, Plos One, and Expert Systems in the past. He is a senior member of IEEE. He is Founding Chair of IEEE Rourkela Subsection and IEEE Kolkata Section Biometrics Council Chapter. He is a distinguished lecturer of IEEE Systems Council for 2023-2025 and distinguished lecturer of IEEE Women in Engineering for 2024-2025. He has published widely in more than 100 journals and conferences.



Houbing Song (M'12–SM'14) is currently with the Department of Electrical Engineering and Computer Science, Embry-Riddle Aeronautical University, Daytona Beach, FL, where he is serving as an Assistant Professor and the Director of the Security and Optimization for Networked Globe Laboratory (SONG Lab, www.SONGLab.us). He has served as an Associate Technical Editor for IEEE Communications Magazine (2017-present), an Associate Editor for IEEE Internet of Things Journal (2020-present) and IEEE Journal on Miniaturization for Air and Space Systems (J-MASS) (2020-present), and a Guest Editor for IEEE Journal on Selected Areas in Communications (J-SAC), IEEE Internet of Things Journal, IEEE Network, IEEE Transactions on Industrial Informatics, IEEE Sensors Journal, IEEE Transactions on Intelligent Transportation Systems, and IEEE Journal of Biomedical and Health Informatics. His research has been featured by popular news media outlets, including IEEE GlobalSpec's Engineering360, Association for Unmanned Vehicle Systems International (AUVSI), Fox News, USA Today, U.S. News & World Report, Forbes, The Washington Times, WFTV, and New Atlas. Dr. Song is a senior member of ACM and an ACM Distinguished Speaker.