Distributed and Parallel Metaheuristic-based Algorithms for Online Virtual Resource Allocation

by

Khoa Nguyen, **M.Sc.**

A dissertation submitted to the
Faculty of Graduate and Postdoctoral Affairs
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario
September, 2021

Abstract

In this dissertation, we propose a metaheuristic Genetic Algorithm (GA) adopted in a distributed and parallel implementation, providing a fast resource allocation approach while maintaining a good performance efficiency.

In the first place, we study the role of virtual link mapping stage in online virtual network embedding (VNE) problem where virtual node mapping (VNoM) and virtual link mapping (VLiM) are conducted separately. We propose several efficient GA-based algorithms driven by nonlinear fitness functions for VLiM stage. Extensive simulation results reveal that our proposed algorithms are not only significantly faster than many existing VNE algorithms, but also better than those in performance.

Although uncoordinated VNoM and VLiM stages can simplified an algorithmic implementation, a lack of coordination between them might result in low embedding outcomes. To fill this gap, in the second part, we propose a new approach that jointly coordinates virtual node and link mappings in a single stage, where the virtual link mapping is based on several efficient path searching methods. Moreover, a novel heuristic conciliation mechanism is presented to handle a possible set of infeasible link mappings. Our proposed algorithms outperform state-of-the-art VNE approaches in all performance metrics we adopted.

In the last part of this dissertation, we propose a collaborative edge computing framework empowered by parked vehicles (PVs) to efficiently handle online computational tasks during peak hours. We formulate the online task offloading as a binary integer programming (BIP) problem aimed at minimizing offloading cost while maximizing accumulative rewards of PVs by sharing their idle resources. To solve the problems of time complexity and scalability of BIP, a heuristic algorithm, namely M&M, and a distributed and parallel GA-based algorithm are introduced. They are then compared with several heuristic algorithms to demonstrate their efficiency on different sizes of generic parking lots.

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Prof. Changcheng Huang, for his support, guidance, and encouragement. It is my privilege to have a great opportunity to work with him. His invaluable advice and patient guidance help me how to conduct a scientific research, and explore new research topics independently. He consistently motivates me to do my best, or encourages me when I have a challenging time during research. His profound supervision pushes me beyond my expectations. I would have never been able to accomplish my dissertation without his tremendous support.

During the long journey of a PhD student, it has been a great pleasure to work with faculty, staff, alumni, colleagues and students at Carleton University. I would like to thank Qiao Lu, Steve Drew, Pei Jin, Hieu Le and other colleagues in laboratory for their support, precious advice and stimulating discussions on my work.

Finally, I would like to appreciate the unconditional support, patience and encouragement from my wife Trang Pham, my daughter Helen Nguyen, our families, Tung Pham, Tin Vo and other friends throughout my PhD journey.

Table of Contents

Abstract ii

Acknowledgments iii

Table of Contents iv

List of Tables viii

List of Figures ix

List of Abbreviations xi

List of Symbols xiv

1 Introduction 1

1.1 Background 1
1.2 Research Motivation 5
1.3 Research Objectives 6
1.4 Thesis Contributions 7
1.5 Thesis Outline 9
1.6 List of Publications 10

2 State of The Art 12

2.1 Virtual Network Embedding Problem 12
2.2 VNE Problem Description and Formulation 14
2.2.1 Network Virtualization Model for VNE 14
2.2.2 Network Entities and VNE Description 16
2.2.3 Main Objectives and Performance Metrics 18
2.2.4 Problem Decomposition and Coordination 19

2.3 Literature Review . . . 21
2.3.1 Exact Methods . . . 22
2.3.2 Heuristic Algorithms . . . 24
2.3.3 Metaheuristic Algorithms . . . 29
2.3.4 Machine Learning Approaches . . . 34
2.4 Distributed and Parallel Computing Framework . . . 43
2.4.1 Introduction . . . 43
2.4.2 Principles of Distributed and Parallel Computing . . . 44
2.4.3 Metaheuristic Genetic Algorithm . . . 45
2.5 Chapter Summary . . . 48

3 Distributed and Parallel GA-based Algorithms 52
3.1 Introduction . . . 52
3.2 Network Model and Problem Descriptions . . . 55
3.2.1 Virtual Network Assignment . . . 55
3.2.2 Performance Metrics . . . 56
3.2.3 Objectives . . . 58
3.3 Distributed and Parallel GA-based Operation Scheme . . . 59
3.3.1 Our Proposed Computing Paradigm . . . 59
3.3.2 Execution Time Analysis . . . 61
3.4 An Intelligent Parallel Algorithm for Online VNE . . . 63
3.4.1 Node Mapping Algorithm . . . 64
3.4.2 Genetic Representation . . . 65
3.4.3 Initial Path Pool Generation . . . 66
3.4.4 Working Nodes . . . 66
3.4.5 Solution Sorting and Terminations . . . 69
3.4.6 Synchronization and VNR Allocation . . . 69
3.4.7 Simulation Setup . . . 70
3.4.8 Compared Methods . . . 71
3.4.9 Execution Time Analysis . . . 72
3.4.10 Performance Results . . . 75
3.5 Genetic Algorithm with An Enhance Fitness Function for VNE Problem 76
3.5.1 Introduction and Motivations . . . 76
3.5.2 An Enhanced Fitness Function . . . 77
3.5.3 Evaluation and Numerical Results . . . 79

3.6 Rethinking Virtual Link Mapping in Network Virtualization 83
3.6.1 Introduction and Motivations 83
3.6.2 Elastic Crossover for Genetic Algorithm 84
3.6.3 Performance Evaluation 87
3.7 Efficient VNE with Node Ranking and Intelligent Link Mapping . . . 91
3.7.1 Introduction and Motivations 91
3.7.2 Node Ranking for VNoM and Distributed Parallel GA-based Algorithm for VLiM 92
3.7.3 Evaluation and Numerical Results 95
3.8 Chapter Summary 97

4 Joint Node-Link Algorithms with Conciliation 99
4.1 Introduction 99
4.2 Our Proposed Joint Node and Link Mapping with Conciliation Strategy 100
4.2.1 Node Mapping Algorithm 101
4.2.2 Link Mapping Algorithms 101
4.2.3 Conciliation Construction 104
4.2.4 Working Nodes 109
4.2.5 Sorting and Terminations 113
4.2.6 Synchronization and VNR Allocation 113
4.3 Evaluation and Numerical Results 113
4.3.1 Simulation Setup 114
4.3.2 Joint Node-Link Mapping with Path Ranking Method 115
4.3.3 Joint Node-Link Mapping with Several Path Searching Algorithms 117
4.4 Chapter Summary 120

5 Collaborative Edge Computing Framework 122
5.1 Introduction and Motivation 122
5.2 Related Work 126
5.3 Problem Formulation 127
5.3.1 Offloading Model 127
5.3.2 System Model 129
5.4 Simulation Setup 134
5.5 M&M Heuristic Algorithm 135
5.5.1 Performance Results 137

5.6 Metaheuristic Genetic Algorithm 139
5.6.1 Genetic Representation and Selection 140
5.6.2 The Processes of Evolution 141
5.6.3 Terminations and Synchronization 143
5.6.4 Compared Algorithms 144
5.6.5 Performance Results 145
5.7 Chapter Summary 148

6 Conclusions and Future Work **150**
6.1 Conclusions 150
6.2 Future Work 153

List of References **154**

List of Tables

2.1 Taxonomy of VNE approaches 43
3.1 *Compared Algorithms* 72
3.2 *Compared Algorithms* 79
3.3 *Compared Algorithms* 87
5.1 Simulation Parameter Settings 136

List of Figures

2.1 *Towards the Future Internet model [1]* 15
3.1 Parallel distributed operation framework 60
3.2 *Parallel operation scheme* 64
3.3 *An example of a VN mapped into a substrate network* 65
3.4 *An example of crossover and mutation operators* 69
3.5 **(a)** VNR Acceptance Ratio **(b)** Average generated revenue 73
3.6 **(a)** Average cost **(b)** Average R/C ratio 73
3.7 Average substrate link consumption 74
3.8 **(a)** Time distribution of $\mathcal{X}_t$ **(b)** Execution time over algorithms 74
3.9 **(a)** VNR acceptance ratio **(b)** Average generated revenue . . . 80
3.10 **(a)** Average embedding cost **(b)** Revenue/cost ratio 81
3.11 **(a)** Average node utilization **(b)** Average path length . . 81
3.12 **(a)** Average link utilization **(b)** Average remaining bandwidth 82
3.13 **(a)** Time distribution of $\mathcal{X}_t$ **(b)** Execution time over algorithms 82
3.14 An example of the elastic crossover operator 85
3.15 **(a)** VNR Acceptance ratio **(b)** Average generated revenue 88
3.16 **(a)** Average cost **(b)** Average path length 88
3.17 **(a)** Average link utilization **(b)** Average remaining bandwidth . . . 89
3.18 Execution time over different algorithms 89
3.19 **(a)** VNR Acceptance ratio **(b)** Revenue/Cost ratio 96
3.20 **(a)** Average remaining bandwidth **(b)** Execution time over algorithms 96
4.1 *Parallel operation scheme* 100
4.2 *Examples of infeasible virtual link mappings* 105
4.3 *Examples of Conciliation and remapping* 109
4.4 *Crossover operation* 110
4.5 *Mutation operation* 112
4.6 **(a)** Avg. acceptance ratio **(b)** Avg. R/C ratio **(c)** Avg. node utilization 114

4.7 **(a)** Average link utilization **(b)** Average path length **(c)** Average delay 114
4.8 **(a)** Average acceptance Ratio **(b)** Average revenue to cost ratio 116
4.9 **(a)** Average generated revenue **(b)** Average embedding cost . . . 116
4.10 **(a)** Average node utilization **(b)** Average link utilization 117
4.11 **(a)** Average delay **(b)** A comparison of execution time between the proposed algorithms 118
5.1 Edge computing architecture integrated with parked vehicles enabled by Kubernetes. 124
5.2 An example of task scheduling 128
5.3 *Performance evaluation* 138
5.4 *Parallel operation scheme* 139
5.5 (a) Acceptance ratio (b) Average costs between architectures 144
5.6 (a) Average offloading cost (b) Average utility . . . 144
5.7 Acceptance ratio towards PV availability 145
5.8 EdgePV's evaluation on different sizes: (a) Acceptance ratio (b) Average cost (c) Average utility 146
5.9 Acceptance ratio towards PV availability with 100 PVs 146
5.10 Average execution time with sequential and parallel operations 147

List of Abbreviations

ACO	Ant Colony Optimization
AI	Artificial Intelligence
BFS	Breadth-First Search
BS	Base Station
CCS	Computation, Communication and Storage
DNA	Deoxyribonucleic Acid
EC	Evolutionary Computation
FA	Firefly Algorithm
FF	Fitness Function
GA	Genetic Algorithm
HS	Harmony Search
IaaS	Infrastructure as a Service
IDC	Internet Data Center
ILP	Integer Linear Programming
InP	Infrastructure Provider
IoE	Internet of Everything
IoT	internet of Things
IoV	Internet of Vehicle
ISP	internet Service Provider

KPI	Key Performance Indicator
LP	Linear Programming
MCF	Multicommodity Flow
MEC	Mobile Edge Computing
MGF	Moment Generating Function
MIP	Mixed Integer Programming
ML	Machine Learning
NaaS	Network as a Service
NFV	Network Function Virtualization
NIG	Normal Inverse Gaussian
NLOS	Non-line of Sight
NV	Network Virtualization
OFDMA	Orthogonal Frequency-Division Multiple Access
PaaS	Platform as a Service
PN	Physical Network
PSO	Partial Swarm Optimization
PV	Parked Vehicle
PVEC	Parked Vehicle Edge Computing
QoS	Quality of Service
RL	Reinforcement Learning
SaaS	Software as a Service
SDN	Software Defined Networking
SINR	Signal-to-Interference-plus-Noise Ratio
SN	Substrate Network

SP	Service Provider
VC	Vehicle Cloud
VDC	Virtual Data Center
VEC	Vehicle Edge Computing
VFC	Vehicle Fog Computing
VLiM	Virtual Link Mapping
VM	Virtual Machine
VN	Virtual Network
VNE	Virtual Network Embedding
VNEP	Virtual Network Embedding Problem
VNF	Virtual Network Function
VNO	Virtual Network Operator
VNoM	Virtual Node Mapping
VNP	Virtual Network Provider
VNR	Virtual Network Request

List of Symbols

Chapter 3 and chapter 4:

$G^s = (N^s, L^s)$	where N^s is the set of substrate nodes and L^s is the set of substrate links
n^s	Substrate node
$loc(n^s)$	Location of each substrate node $n^s \in N^s$
$c\,(n^s)$	CPU capacity of a substrate node
l^s	Substrate link
$b\,(l^s)$	Bandwidth capacity of a substrate link $l^s \in L^s$
i^{th}	Arriving VN request
$G_i^v = (N_i^v, L_i^v)$	where N_i^v and L_i^v are the set of virtual nodes and links of the i^{th} VN request
n^v	Virtual node
$c(n_i^v)$	CPU requirement of a virtual node $n_i^v \in N_i^v$
$b\,(l_i^v)$	Bandwidth requirement of a virtual edge $l_i^v(s_i^v, d_i^v) \in L_i^v$ between the corresponding virtual source s_i^v and destination nodes d_i^v
$D(n_i^v)$	Each VNR has a mapping radius expressing how far a virtual node n_i^v can be placed from the location identified by $loc(n_i^v)$
$\mathcal{A}_{\mathcal{N}} : N_i^v \rightarrow N^s$	Function that represents a virtual node from the same VN request can be embedded onto a single substrate node
$n^v \rightarrow n^s$	Indicates the virtual node n^v hosted on the substrate node n^s

$\mathcal{D}(i^s, j^d)$	Distance between two nodes i^s and j^d
$R_N(n^s)$	Residual CPU capacity of a substrate node
$\mathcal{A}_\mathcal{L} : L_i^v \rightarrow L^s$	Function that represents a virtual link can be embedded onto a substrate path (unsplittable) with one or more substrate links. with $l_i^v = (s_i^v, d_i^v) \in L_i^v$
$\mathcal{E}^s(\mathcal{A}_\mathcal{L}(l_i^v))$	A possible set of all physical paths from source node $\mathcal{A}_\mathcal{N}(s_i^v)$ to destination node $\mathcal{A}_\mathcal{N}(d_i^v)$
$R_L(e^s)$	Available bandwidth of the physical path $e^s \in \mathcal{E}^s$
$R_L(l^s)$	Residual capacity of a physical link
$R(G_i^v)$	Revenue of the i^{th}
A_c^τ	Ratio between the number of accepted VNRs over arrived VNRs during the interval time τ
$\xi^a(\tau)$	Successfully embedded VNRs
$\xi(\tau)$	Number of arrived VNRs
$C(G_i^v)$	Total substrate network resources that have been already allocated to the i^{th} virtual network
$f_{l^s}^{l_i^v}$	Bandwidth of the substrate link l^s allocated to the virtual link l_i^v
$\mathcal{RC}(G_i^v)$	Ratio of average embedding revenue over average embedding cost
$\mathcal{U}_L(L^s)$	Link utilization reflects the distribution of network loads throughout the SN
$\mathcal{R}_m(L^s)$	Remaining bandwidth of a substrate network
$\mathcal{D}_E$	Euclidean Distance between two nodes
$\Theta(l^s, l^v)$	Attraction strength of a link $l^s \in e^s : \forall e^s \in \mathcal{E}^s(\mathcal{A}_\mathcal{L}(l_i^v))$
$\mathcal{F}(\mathcal{S}_z)$	Fitness function of z^{th} feasible solution

Chapter 5:

$G = (N, L)$	Edge network where N is a set of worker nodes, whereas L is the corresponding link
$c(k)$	CPU requirement of task k
$m(k)$	Memory requirement of task k
$b(k)$	Bandwidth requirement of task k
$t_m(k)$	Tolerable maximum latency of task k
$\eth(k)$	A number of required replicas of task k
k_j	a j^{th} replica of the task $k \in K$
n_i	A worker node $n_i \in N$
$C(n_i)$	CPU capacity of $n_i \in N$
$C_{c\|e\|p}$	CPU capacity of Cloud | Edge | PV
$M(n_i)$	Memory capacity of $n_i \in N$
$M_{c\|e\|p}$	Memory capacity of Cloud | Edge | PV
B_c	Bandwidth capacity of the link between Cloud and Edge
K	A set of tasks offloaded to network successfully
$K_{c\|e\|p}$	A set of tasks offloaded to cloud or edge or PVs successfully
$R_C^u(n_i)$	The remaining CPU capacity of a worker node $n_i \in N$
$R_M^u(n_i)$	The remaining memory capacity of a worker node $n_i \in N$
$t_{c\|e\|p}$	Cloud or Edge or PVs offloading latency
ξ_c	Server transmission rate
ξ_p	PV transmission rate
χ_{k_j}	Input data size
f_k	CPU cycles
d	Distance between cloud and edge
$d_{bs,p}$	Distance between BS and a PV

v	Speed of light
T_h	Managing time of a task
B_p	Channel bandwidth
P_{TX}	Transmission power of BS
I	Inter-cell interference
N_0	Gaussian noise power
$d_{bs,p}^{-\sigma}$	Path loss
σ	Path loss exponent
$\Xi_{C_{c\|e\|p}}$	CPU cost at the Cloud | Edge | PV
$\Xi_{M_{c\|e\|p}}$	Memory cost at the Cloud | Edge | PV
$\Xi_{B_{c\|p}}$	Bandwidth cost at the Cloud | PV
$W_{C_{c\|e\|p}}$	CPU weight parameter at the Cloud | Edge | PV
$W_{M_{c\|e\|p}}$	Memory weight parameter at the Cloud | Edge | PV
$W_{B_{c\|p}}$	Bandwidth weight parameter at the Cloud | PV
$\Xi_{k_j}^{c\|e\|p}$	Total cost of offloading a task replica to the Cloud | Edge | PV
δ	A small positive number to avoid dividing to zero
E_p	Energy consumption at a PV
e_p	A coefficient
ϵ	An energy coefficient
ς	Energy cost coefficient
φ^p	Rewards by accepting a task replica at a PV
ϖ^p	Utility
ρ	A coefficient of energy price
r_p^c	Unit price of CPU resources

r_p^m	Unit price of memory resources
$\mathcal{A}_{k_j}^{c\|e\|p}$	Binary variables if k_j deployed at Cloud | Edge | PV
η	Damping factor within (0,1)
α	Maximum proportion of task replicas assigned in each PV

Chapter 1

Introduction

1.1 Background

The contemporary network infrastructures have become more complex, that requires new efficient mechanisms or methods to facilitate the deployment and management of networking facility. Virtualization is widely acknowledged as an enabler to extend the flexibility in network resource management as well as to enhance network efficiency. Network virtualization (NV) is a presentation form of this key concept on network components including nodes and links. Over the past decade, NV has received an intensive attention from both industry and academia. Due to the increase of computation power of commodity hardware, virtualization techniques have been revised and implemented on commercial off-the-shelf hardware to achieve an extended flexibility on resource management. These techniques applied to network resources could address the perceived ossification of current Internet infrastructure [2, 3].

Recently, NV has been exploited in different research fields. Data center virtualization bases on virtualization techniques to partition available network resources, being shared amongst several different users. A Virtual Data Center (VDC) is considered as a logical instance of a virtualized data center (e.g., Virtual Machines (VMs), virtual routers or switches) that includes a subset of physical data center resources. Partly or all of hardware of a data center (e.g., servers, routers, switches and links) can be virtualized. Virtualized data center brings an entire view of the network system to the corresponding infrastructure providers, hence simplifying resource allocation and enabling failure mitigation efficiently [4]. Moreover, NV largely benefits the cutting-edge technologies such as Software Defined Networking (SDN) [5] and Network Function Virtualization (NFV) [6] by its dynamic network segmentation and utilization. These

technological trends would become paramount in future. NV is also an emerging paradigm for approaching prospective network architectures such as virtualized 5G network [7] and smart Internet of Things (IoT) networks [8]. NV retains diversified services and applications through the sliceable management of the existing substrate resources. NV also allows a seamless resource sharing amongst multiple VNs, enabling the isolated coexistence of several VNs on a single SN. This considerable advantage averts an unexpected expansion of the existing infrastructure and enhances network efficiency.

The innovative idea of NV is to decouple a typical Internet Service Provider (ISP) into two different business components: Infrastructure Providers (InPs) and Service Providers (SPs). The former manages network resources and sustains the desired level of Quality of Service (QoS) at network layer whereas the latter is responsible for implementing network protocols and offering end-to-end services. Typically, SPs convert services/applications into Virtual Networks (VNs), and then convey them to InPs under a composition form of Virtual Network Requests (VNRs). InPs attempt to map the corresponding VNRs comprising a set of virtual nodes interconnected with virtual links constructing an explicit topology on top of a physical infrastructure. This procedure is carried out by an optimization process that is required to meet various rigorous resource constraints. Embedding VNs onto a SN efficiently is the major challenge of the resource allocation problem in NV, which is widely known as Virtual Network Embedding (VNE). Mapping multiple VNs onto a physical network which supports either splittable or unsplittable-path configuration is the dominant topic in several research fields such as SDN, NFV, future Edge Cloud, virtualized 5G and IoT networks. Despite producing better resource utilization with splittable-enabled embedding in theory, the splittable-enabled mapping approaches could generate abundant overheads for consistently maintaining network states. These solutions probably induce the problem of out-of-order package delivery with extra latency that could not serve sensitive-delay applications. Splittable-enabled strategy also increases the complexity of algorithmic implementations, but it cannot guarantee its efficiency.

VNE is widely known as a generalization of second stage of Network Function Virtualization Resource Allocation (NFV-RA), namely Virtual Network Function Forwarding Graph (VNF-FG) embedding since VNE is in the same problem domain with NFV-RA in the aspect that the goal of both problems is to efficiently allocate VNRs on the top of physical infrastructures [9]. NFV-FGE is also the central problem

of NFV-RA accompanying with Virtual Network Functions (VNFs) chain composition (known as service function chaining) and VNFs scheduling [9]. In fact, VNE is assumed to be even more complicated than NFV-FGE in some specific cases and topologies [10], [11].

Optimal resource allocation is indispensable to guarantee customized end-to-end services to end users with respect to desired objectives. However, optimizing resource allocations for online VNE applications is never an easy task. In real world, we tend to solve the problems of online virtual resource allocation where online VNRs keep arriving the network so the time is extremely critical. By human nature, users need fast and immediate responses instead of waiting for a long time. Moreover, online virtual resource allocation is more challenging than the online counterparts. Unfortunately, there are no actual optimal allocation solutions for online VNRs due to the fact that designing an optimization model that is able to take the future factor into account is extremely complicated since future is associated with dynamics and uncertainty. We do not have the data of future VNRs, so approaching optimal solutions for future demands seems impractical. Thus, we can only improve the allocation results by approaching more efficient VNE algorithms, but we cannot indeed enhance their optimality.

VNE problem indeed addresses the allocation of VNs in both nodes and links. Due to its complexity, this process can be decoupled into two sub-problems: Virtual Node Mapping (VNoM) and Virtual Link Mapping (VLiM). VNoM allocates virtual nodes onto substrate nodes whereas VLiM maps virtual links, that connect virtual nodes, to physical paths in substrate networks (SNs) [1]. VNE is broadly recognized as $\mathcal{NP}$-Hard either towards VNoM or VLiM stage [9], [12]. A large number of VNE solutions have been proposed over the past decade, solving the optimization problems of resource allocations in virtualization environments. In fact, some formulated optimization models (e.g. Integer Linear Programming) are usually recommended for achieving optimal VNE solutions; unfortunately, these solutions encounter many challenges since the proposed models have to confront with scalability, intricate deployment as well as time complexity. Thus, exact solutions are not apt to deal with online VNE problem. As a result, the majority of VNE studies have concentrated on efficient designs of heuristics or metaheuristics to alleviate the aforementioned hurdles of the mathematical models.

Till now, most VNE approaches have been proposed to solve VNoM and VLiM

phases uncoordinately.A large number of VNE solutions merely focus on VNoM stage pursuing efficient node mapping mechanisms while mainly relying on the shortest path methods (e.g. k-shortest path) or multicommodity flow (MCF) mechanism for VLiM stage [13]. Additionally, the shortest path method is widely known as the fastest VLiM mechanism in VNE. These options obviously limit the possible selections for virtual network link embedding. Greedy node mapping following the shortest path method is ubiquitously acknowledged as the fastest approach in speed due to its simplicity, even it suffers a sub-optimal performance. However, this approach is highly practical due to its rapid embedding speed which is a decisive factor in online dynamic VNE environment.

Moreover, a decoupling between VNoM and VLiM can facilitate the complexity of algorithmic deployment, but this common approach is most likely to sacrifice the embedding efficiency. A lack of coordination between VNoM and VLim stages would lead to low acceptance ratio and correspondingly low network revenue as well as higher mapping cost in average. The reason is that although the VNoM stage can figure out highly promising substrate nodes for mapping virtual nodes in a VNR, we may fall into an unexpected situation that there are no feasible physical paths in the SN that have sufficient remaining capacity to connect one or more pairs of those virtual nodes together with respect to node mappings in previous stage. Then, the corresponding VNR is certainly rejected. Indeed, those rejected VNRs can be reassessed after the link mapping stage or be put back into the queue and waiting until the network is idle. These approaches are inefficient and unpractical since they could violate the stringent delay constraint of online VNRs. Network resources utilized for processing such failed VNRs in the past are consequently wasted. As a result, it is required efficient VNE algorithms that can consider node-link mappings coordinately. If node mappings are inappropriate leading to several failures of link mappings, we need an efficient remapping approach that allows to minimize the number of remapped nodes and links. Designing an online VNE solution that meets these indispensable demands has been still infancy.

Accordingly, there is a trade-off of the existing VNE approaches between speed and performance efficiency. While complicated optimization models are somehow able to produce optimal solutions, but they suffer the problem of time complexity. In contrast, heuristic approaches can possibly assure the rapid embedding speed, but they have to scarify the efficiency by accepting sub-optimal solutions.

1.2 Research Motivation

In dynamic environment of online VNE, time is critical when a large number of online VNRs keep arriving in the network. If VNE approaches are not fast enough, their embedding time may not meet tolerable delay requirements of users or the users' experience will be significantly affected due to high processing delay. Thus, practical VNE algorithms are required to allocate resources as fast as possible while maintaining performance efficiency. Unfortunately, current VNE solutions could not completely achieve this goal.

Recently, we have witnessed a widespread adoption of Machine Learning (ML) techniques, a subset of Artificial Intelligence (AI), in diverse domains at an unprecedented level. The most emerging research in ML algorithms recently is Reinforcement Learning (RL), that has been successfully applied in various optimization networking domains [14–33]. RL algorithms provide a promising approach to solving many problems. However, they face some fundamental challenges when being applied to online VNE applications. The challenges come from the fact that RL algorithms are based on parametric models, which are highly dependent on stationary environments. A specific Markov Decision Process (MDP) model has to be trained for a specific environment. A large number of environment scenarios mean a large number of models. When environments are limited to few stationary scenarios, RL algorithms can work very effectively. However, when dealing with very dynamic environments like online VNE, they suffer from the curse of dimensionality that are very hard to solve.

Genetic Algorithm (GA), a mature metaheuristic algorithm motivated by the Darwin evolution principle with natural selection base, is one of the most popular population-based metaheuristic methods in Evolutionary Computation (EC). GA is suitable for solving multi-objective linear or non-linear optimization problems due to its simplicity and ease of implementation. By balancing between exploration and exploitation, GA provides a set of feasible solutions that can be evolved over genetic operations driven by a fitness function where several network attributes (e.g., residual capacity) can be simply integrated. Another fundamental feature of GA algorithm is parallelizability since it is widely recognized as a parallel searching mechanism [34] with non-mutual dependency amongst several exclusively feasible solutions. Additionally, cloud computing built on massive datacenters has become the most popular computing paradigm for low-cost computation services. However, it poses a significant challenge of utilizing network resources efficiently. Several

programming models (e.g., MapReduce [35]) are successfully developed to deploy large-scale parallel computing with small extra cost. Hence, a GA-based algorithm deployed in a distributed and parallel manner (e.g., cloud) can guarantee a fast embedding speed while maintaining a comparative efficiency.

In fact, our GA-based approach shares an analogy with RL algorithms with a regard that both algorithms interact with environments through an iterative action-reward process. GA is commonly considered as a scalable alternative to RL algorithms with competitive performance results [20] and [24]. Unlike RL algorithms, our VNE solution is model-free approach which does not depend on any particular models, and this approach is applicable to either non-stationary or stationary environments without any pre-training. Our proposed approach not only meets the stringent delay requirements for online VNE applications by adopting the distributed and parallel implementations, but also attains better performances than several existing VNE algorithms. As a result, we believe our approach is fast, efficient, and practical.

1.3 Research Objectives

Towards online VNE, exact methods with optimization models encounter the time complexity problem which is inapplicable to online mappings. On the other hand, heuristic algorithms can patch this issue by handling online VNRs fast, but they suffer sub-optimal solutions due to their simplification. There is a dilemma about prioritizing either fast embedding speed or performance efficiency.

To fill this gap and approach an efficient embedding algorithm that can be practically applied to online VNE applications, we propose a distributed and parallel GA-based algorithm for dealing with resource allocation problems, mainly targeting a fast embedding speed while producing highly comparable performance results. Our approach is verified its efficiency in various major domains.

First, we revise the role of virtual link embedding stage in the VNE process which has been usually overlooked in a majority of research papers. We assume that VLiM plays a critical contribution to an efficient resource allocation in network virtualization. As such, an efficient link mapping algorithm can achieve great embedding efficiency, even if a simple node mapping method is used. Furthermore, our approach is faster than the well-known fastest VNE approach while performing better than several VNE algorithms in performance, making it very practical for online VNE applications. To

fix the gap between node and link mapping stages that have been usually conducted separately, possibly producing low embedding outcomes, we then propose a joint node-link mapping solution to handle this problem as well as a novel heuristic algorithm that allows to keep the number of remapped nodes and links minimal. Finally, we investigate our proposal in a collaborative computing framework where cloud, edge and parked vehicles are incorporated to tackle online task offloading problem during peak hours.

1.4 Thesis Contributions

In the first part of this dissertation, we investigate the role of VLiM stage in online VNE problem by proposing several efficient embedding algorithms that are based on distributed and parallel GA-based algorithms to handle the VLiM, whereas the VNoM is relied on either a simple greedy method or a node ranking mechanism. We have proved that VLiM plays an essential role in dealing with online VNE problem, which is usually abandoned in most of research papers where VLiM is commonly confided to the shortest path method due to its simplicity and rapid speed. Our proposed algorithms for VLiM not only enhance the VNs embedding performance, but also perform absolute faster than the fastest and most popular heuristic VNE algorithm for more than 40% by adopting distributed and parallel implementations. That makes our approaches highly applicable to online VNE applications.

In particular, we define a potential chromosome represents a link mapping solution of all virtual link requests of a given VNR; correspondingly, each gene within a chromosome is the mapping solution of a virtual link. Moreover, we design a new crossover scheme applicable to VLiM, namely, elastic crossover, where the random number of genes are exchanged between the selected parents to generate new generations. A new mutation that operates a path-based gene modification on the selected chromosome is also presented. Additionally, we introduce multiple efficient non-linear fitness functions that are based on several network resource factors such as cost, hop-count, residual bandwidth, propagation delay and a novel attraction strength. These fitness functions can drive GA algorithms to better performance efficiency, and somehow enable a better prediction about future (e.g., through residual resource capacity). For the first time, an attraction strength is proposed as a factor in a fitness function for online VNE applications. Our solutions perform better than several existing VNE algorithms

in both performance, and speed in various evaluation metrics. It can be concluded that our distributed and parallel GA-based algorithms have successfully achieved the expected goals including fast, efficient and practical. This is the most novel and impressive contribution in this dissertation.

The next contribution of this dissertation is that we resolve the contradiction caused by handling VNoM and VLiM uncoordiantely, which leads to low embedding results. In essence, we propose a new joint node and link mapping approach based on GA algorithm to efficiently embed online VNRs, where VLiM is relied on different link mapping algorithms including random path searching, sequential single path searching, path ranking and finally k-shortest path method. The idea is that virtual node and link mappings are conducted in a coordinated manner so when node mappings are changed, the link mappings are altered accordingly. To handle a possible set of infeasible link mappings due to improper node mappings, a novel heuristic conciliation mechanism aimed at keeping the number of remapped nodes and links minimal is ultimately introduced. Extensive simulation results indicate that our new approaches outperform state-of-the-art VNE algorithms in all critical performance metrics we adopted for up to 77.83%.

Furthermore, we would like to demonstrate that our proposed distributed and parallel GA-based algorithm can be applicable to a new research field; specifically, the edge computing which is recently one of the most emerged research area in academia and industry. In last contribution, we propose a collaborative computing framework that integrates cloud, edge, and parked vehicles (PVs) into a single unified infrastructure to cope with the online task offloading problem during peak hours. To guarantee service continuity, we advocate a container orchestration framework based on Kubernetes platform to tackle the uncertainty of parking duration of PVs. When a PV node fails, Kubernetes automatically reschedules the requested task replicas into another healthy node. Moreover, we analytically formulate the task-offloading problem as Binary Integer Programming (BIP) subject to minimizing offloading cost while maximizing accumulative rewards. Besides GA algorithm proposed for tackling time complexity and scalability of BIP, we also propose a heuristic algorithm, namely M&M, to dynamically address online heterogeneous demands. Moreover, we consider an accumulative incentives model, in which the PV owners can profit from selling idle computing resources during the parking time. We compare the proposed algorithms with a set of existing heuristics on different sizes of generic parking lots. Our simulation

results show that the proposed PV-enabled framework not only extends the capacity of the existing computing infrastructure by accepting more online tasks arrived, but also reduces average offloading cost for at least 40%. Furthermore, we quantify the availability of PVs associated with task acceptance ratios. These information can become crucial for network planners who would like obtain the desired network service goals.

1.5 Thesis Outline

This dissertation proposes an efficient distributed and parallel GA-based approach for solving online virtual resource allocation problem. Accordingly, Chapter 2 introduces virtual network allocation problem, its description and formulations. Then, it covers the related work by conducting a literature review in which different VNE approaches are well categorized. A framework that is based on GA algorithm adopted in a distributed and parallel manner to enhance the embedding speed is also provided in this chapter.

Chapter 3 presents the network model and a distributed and parallel GA-based operation scheme as well as its execution time analysis. This chapter introduces several VNE algorithms for VLiM stage. An intelligent GA-based VN embedding algorithm with and without node-ranking method for VNoM and its several enhancements including enhanced fitness functions and a novel crossover operator of GA algorithm for solving online VNE problem are described.

Chapter 4 addresses a conflict between virtual node and virtual link mapping stages, which results in low mapping results. As such, a joint node-link embedding based on GA algorithm in which the VLiM is relied on different path searching methods and a novel heuristic conciliation mechanism are described in details.

Chapter 5 describes the leverage of the proposed distributed and parallel GA-based approach for dealing with online task offloading problem at the edge. As such, the collaborative computing framework and online task offloading formulations with various objectives are detailed. We then propose M&M, a heuristic algorithm, and EdgeGA, a distributed and parallel GA-based algorithm, to solve the problems of time complexity and scalability of BIP. The offloading model with the random mobility behaviors of PVs under random and dynamic task arrivals are also expressed in this chapter.

Finally, Chapter 6 concludes the dissertation and presents our future work.

1.6 List of Publications

1. **K. Nguyen** and C. Huang, "An Intelligent Parallel Algorithm for Online Virtual Network Embedding," in *2019 International Conference on Computer, Information and Telecommunication Systems (CITS)*, Beijing, China, Aug. 2019, pp. 1-5, doi: 10.1109/CITS.2019.8862072.

2. **K. Nguyen** and C. Huang, "Distributed parallel genetic algorithm for online virtual network embedding," *Wiley International Journal of Communication Systems*, 23 Dec 2020, pp. e4691, doi: 10.1002/dac.4691.

3. **K. Nguyen**, Q. Lu and C. Huang, "Rethinking Virtual Link Mapping in Network Virtualization," *2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall)*, 2020, pp. 1-5, doi: 10.1109/VTC2020-Fall49728.2020.9348799.

4. **K. Nguyen**, Q. Lu and C. Huang, "Efficient Virtual Network Embedding with Node Ranking and Intelligent Link Mapping," *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*, 2020, pp. 1-5, doi: 10.1109/CloudNet51028.2020.9335801.

5. **K. Nguyen**, Qiao Lu and C. Huang, "Joint Node-Link Embedding Algorithm based on Genetic Algorithm in Virtualization Environment," *2021 IEEE 94th Vehicular Technology Conference (VTC2021-Fall)*, 2021, pp. 1-5. (accepted for publication).

6. **K. Nguyen** and C. Huang, "Towards Adaptive Joint Node and Link Mapping Algorithm for Embedding Virtual Networks: A Conciliation Strategy," in *IEEE Transactions on Network and Service Management*, 2021 (submitted).

7. **K. Nguyen**, S. Drew, C. Huang and J. Zhou, "Collaborative Container-based Parked Vehicle Edge Computing Framework for Online Task Offloading," *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*, 2020, pp. 1-6, doi: 10.1109/CloudNet51028.2020.9335809.

8. **K. Nguyen**, S. Drew, C.Huang, J.Zhou, "EdgePV: Collaborative Edge Computing Framework for Task Offloading", *ICC 2021 - 2021 IEEE International Conference on Communications (ICC)*, Montreal, Canada, 2021 (to appear).

9. **K. Nguyen**, S. Drew, C.Huang, J.Zhou, "Parked Vehicles Task Offloading in Edge Computing," in IEEE Access, pp. 01-13, 2021 (submitted).

10. Q. Lu, **K. Nguyen** and C. Huang, Distributed parallel algorithms for online virtual network embedding applications. *Wiley International Journal of Communication Systems*, 24 Jan 2020, pp. e4325, doi: 10.1002/dac.4325.

11. Q. Lu, **K. Nguyen** and C. Huang, "A Novel One-stage Distributed Parallel Embedding for Virtualized Network Environment," in *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Toronto, ON, Oct. 2020, pp. 395-400, doi: 10.1109/SMC42975.2020.9282829.

12. Q. Lu, **K. Nguyen** and C. Huang, "GAONE: A Novel Approach for Online One-stage Virtual Functions Embedding", *Journal of Networking and Network Applications*, 2021 (to appear).

Chapter 2

State of The Art

2.1 Virtual Network Embedding Problem

Embedding VNs onto a shared physical network is the major resource allocation challenge in NV, which is ubiquitously perceived as *Virtual Network Embedding* problem. Optimal allocations of multiple isolated and customized VNs on the underlying substrate resources is significantly important in order to enhance the network utilization and to maximize the number of embedded VNs. VNE involves the allocation of virtual resources in terms of nodes and links, so mapping virtual nodes and links onto a SN raises several primary resource allocation problems. SNs with physical nodes and links have limited resource capacities (e.g., CPU, bandwidth) while online VNRs require heterogeneous resource demands. An attempt to efficiently embed as many VNRs as possible onto a given SN with limited resource capacity is an $\mathcal{NP}$-hard problem. Even a solution of a virtual node mapping is given, the problem of assigning a set of virtual links onto a single/multiple substrate paths efficiently is still $\mathcal{NP}$-hard. Thus, optimal solutions can merely be achieved for small network instances. As a result, the research community usually focus on heuristic or metaheuristic approaches.

In fact, VNE solutions can be classified into six categories including *Static or Dynamic*, *Centralized or Distributed*, and *Concise or Redundant*in literature [1], [39], [40]. Each of them is mutually independent.

1. Static vs Dynamic:

In real-world situations, VNRs dynamically arrive and stay in network for a random duration (online VNE), and VNE algorithms need to address them as they arrive instead of visiting a whole set of VNRs at once. Static VNE solutions do not

consider re-embedding strategies to enhance the mapping performance due to resource fragmentation, any changes of VNs or SNs. In contrast, dynamic approaches attempt to reconfigure the already-mapped VNRs in order to re-optimize network resource allocations.

2. Centralized vs Distributed:

In centralized scheme, one entity performs the VNs mapping, which would simplify the embedding procedure due to the fact that such entity is aware of the global view of the network status. However, this strategy might face a problem of scalability or outage when the network is overwhelmed by a large number of arriving VNRs or the processing entity fails. On the other hand, multiple entities for handling VNRs are deployed in a distributed scheme which can adapt to better network scalability since the workload can be distributed amongst several entities. However, its trade-off is the synchronization overhead when each entity does not have a global view of network state so that less information available at an entity, less efficient embedding would occur. As a result, communication cost and mapping quality should be conscientiously considered. An example of this situation is that multiple InPs in which each InP is responsible for allocating a VN (InterInP VNE) can be considered as a distributed manner.

3. Concise vs Redundant:

In fact, a failure of a physical entity would have an effect on all possible virtual nodes that are running on it, so if VNs that are associated with sensitive-fault tolerance applications are embedded on SNs, redundancy strategy is advisable to embedding solutions that can be utilized as fall-back/backup resources when the primary resources fail. By this definition, concise solutions do not reserve redundant resources while mapping VNs, meaning that they try to exploit as many physical resources as necessary to meet the VNs' demands. Hence, there is no guarantee on embedding solutions in case network failure happens. In contrary to concise approaches, redundant strategies tend to reserve extra resources for mapping VNs, that can be useful in case substrate resources accidentally change. We also need to monitor the substrate entities, detect any possible failures and then activate a reliable and resilient mechanism to recover corresponding changes of the substrate resources. However, if there is no failure occurred, the reserved resources of SNs are certainly wasted. It said that there should be a careful consideration on the trade-off between reliability and mapping costs. In

practise, although substrate nodes and links of a SN might have an equal chance to fail, single link that gets failed would be more complicated than a failure of a substrate node since it is associated with an already-embedded substrate path traversing several physical links meeting bandwidth constraint in all sub-sequence links. Bandwidth constraints ought to be attentively concerned to prevent the overloading issues of the physical link capacity.

2.2 VNE Problem Description and Formulation

This section presents network virtualization model for VNE problem and its description and formulation.

2.2.1 Network Virtualization Model for VNE

As described in 1.1, the original idea of NV is to basically split a traditional Internet Service Provider (ISP) into two different business components. This proposed scheme allows to increase the flexibility, manageability and stability of underlying SN. Recent NV business models have been introduced to decouple ISPs into different roles such as *Infrastructure as a Service* (IaaS), *Software as a Service* (SaaS), *Platform as a Service* (PaaS) and *Network as a Service* (NaaS) business models.

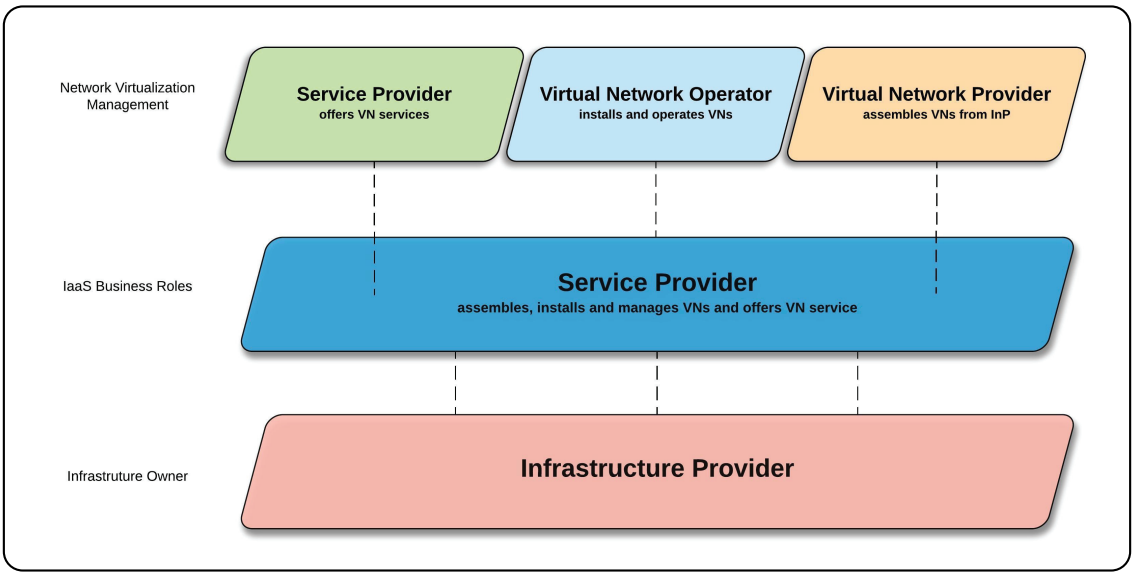


Figure 2.1: *Towards the Future Internet model [1]*

NaaS business model advises the typical ISPs to be divided into InPs and SPs [41]; whereas in [42], the role of SP is further split into multiple customized components: *Virtual Network Provider* (VNP), *Virtual Network Operator* (VNO) and *Service Provider* (SP), targeting a coordination of different network functions and services. Figure 2.1. illustrates a recent Internet business model as well as NV business model for VNE. However, to facilitate the network model and make them applicable to VNE problem, NaaS is recommended as the NV business model for VNE [39]. As such, the NV business model for VNE problem is composed of two direct components: InPs and SPs. InPs are in charge of constructing, managing, maintaining and trading their available substrate resources to multiple SPs. In contrast, SPs concentrate on providing and handling heterogeneous end-to-end services to end users by renting physical resources from multiple InPs. In short, InPs own the network resources and maintain a desired level of Quality of Service (QoS) at the network layer whereas SPs aggregate substrate resources leased from multiple InPs into VNs to provide customized services to end users. Thus, InPs earn revenues from the resources sold to SPs while SPs achieves profits from their active users. This separation of networking and service implementations from the SN infrastructure simplifies the network innovation. In general, a SP converts a service or an application into a VN, then conveys it to an InP under a form of VNR. The underlying InP attempts to embed the corresponding VN, that includes a set of nodes connected via links constituting an explicit topology, onto

its infrastructure through an optimization process with several stringent constraints. Indeed, a virtual node/link can be mapped on a single or several substrate nodes/paths respectively.

To achieve various desired objectives such as maximizing the revenues of SPs and enhancing resource utilization of underlying InPs, efficient VNE algorithms that are able to approach optimal resource allocation solutions are considerably critical to provision end-to-end guaranteed services to contracted users. Hence, SPs should have diverse mapping algorithms/strategies for heterogeneous customized VNs.

2.2.2 Network Entities and VNE Description

This section discuss on primary entities in VNE and then VNE descriptions.

In VNE, there are two main entities comprising substrate network (or physical network) and virtual network (also known as sliceable network [39]). Physical network is often used in tandem with SN in many research papers. In theory, the main difference between two terminologies is that SN allows to share the same physical resources of a single network infrastructure with several virtual networks, enabling an isolated coexistence between them. SN is owned and administrated by an InP, which is composed of multiple physical nodes (e.g. servers, data centers) that are connected by a set of substrate links, forming a physical network topology. It has a finite physical resources (e.g., CPU, storage, bandwidth) and different characteristics (e.g., location, distance). Similarly, a VN includes a set of virtual nodes and another set of virtual links, forming a virtual network topology. Each VN has its specific resource demands (e.g. CPU, storage, delay, location) and varied topology (e.g. star, bus, arbitrary). Moreover, each VN has time attributes including arrival time and duration. All of these features make up a virtual network request. Due to virtualization, multiple VNs can coexist and be isolated on a shared SN. VNs are usually processed successively and mapped onto the shared SN based on their arrival time. Achieving optimal VNE solutions is a critical issue in NV. Due to its complexity, a VNE process can be divided into two separate sub-problems: virtual node mapping (VNoM) and virtual link mapping (VLiM) stages.

1. Virtual Node Mapping Stage:
   Each virtual node from the same VNR is usually assigned to a substrate node in the underlying SN. Allocations of all virtual nodes are characterized by a node mapping function $\mathcal{A}_{\mathcal{N}} : N^v \longrightarrow N^s$ where N^v and N^s indicate a set of virtual

and substrate nodes respectively.
For a VNR such that $\forall n^v \in N^v$ subject to: $\Phi_N(n^v) \leq \Psi_N(\mathcal{A}_\mathcal{N}(n^v))$, where Φ_N and Ψ_N denote demands and node capacity of an element (either SN or VNR) respectively. It means that all virtual node resource requests including resource demands and constraints must be fulfilled in node mapping stage. In VNE aspect, the resource demands of a virtual node normally refer to CPU capacity, storage and node location. In practise, Internet applications' s node resource demands may fluctuate over time within a VN lifetime [39]. Predicting the workload changing of VNs to well serve end users is not an easy task. Consequently, basic node resource demands and their variances can be considered as dynamic resource demands in VNE. For simplification, most VNE models assume that all virtual node resource demands are constant during the VN lifetime.

2. Virtual Link Mapping Stage:
In this stage, an individual virtual link of the given VNR can be mapped onto a single substrate path (unsplittable flow) or multiple substrate paths (splittable flow) between two physical nodes that host the corresponding virtual nodes. A substrate path consists of one or multiple physical links. Similar to the node mapping, the VLiM of a VNR can be presented by link mapping function $\mathcal{A}_\mathcal{L} : L^v \longrightarrow L^s$ for all virtual links. where L^v and L^s indicate a set of virtual and substrate links respectively.
With a virtual link request, if $\mathcal{E}^s(\mathcal{A}_\mathcal{L}(l^v))$ is a possible set of all substrate paths that are able to host l^v, such that $\forall l^v \in L^v$: $\mathcal{A}_\mathcal{L}(l^v) \subseteq \mathcal{E}^s$, subject to: $\Phi_L(l^v) \leq \Psi_L(\mathcal{A}_\mathcal{L}(l^v))$, where Φ_L and Ψ_L denote demands and link capacity of an element (either SN or VNR) respectively. All virtual link resource constraints must be all fulfilled in this stage. Virtual link resource demands normally refer to link bandwidth and link propagation delay. Virtual link resource demands may change during VN lifetime as similar to node mapping constraints, so the same strategy can apply.

VNR is successfully embedded only if both node and link mappings must be passed. Different VNRs may have different resource demands including nodes and links. It is important to acknowledge that virtual link embedding is more challenging than its counterpart due to several rigorous requirements. Specifically, all substrate links of the potential link mapping solutions of a VN link request that a path traverses must have sufficient residual capacity, which may result in bandwidth fragmentation

problem. Moreover, mapping requested virtual links on the underlying shared substrate infrastructure with several stringent constraints is still $\mathcal{NP}$-hard, even for offline mapping [1].

2.2.3 Main Objectives and Performance Metrics

VNE involves in seeking optimal solutions for VNoM and VLiM stages with respect to various objectives. This section expresses the objectives that are usually pursued by most popular VNE approaches.

1- *Maximize the profit of InPs and SPs*:

As described in previous section, an InP in NaaS model is decoupled into InP and SP. Its natural objective is to maximize the economical profit (long-term average revenue) by directly accepting as many VNRs as possible (VN acceptance ratio). To achieve this target, VNE algorithms attempt to minimized network resources of SN to embed VNRs, known as embedding cost. On the other hand, revenues upon renting normally refer to the profits gained from multiple SPs. In short, SPs have to lease physical resources of InPs to provide services to their contracted users. Regarding the profit of SPs, revenues earned are proportional to the number of successfully embedded VNs requested by end users. To maximize the mapped revenues, SPs try to minimize the rented substrate network resources for embedding VNs. It said InPs and SPs both have a mutual goal aimed at using the physical resources wisely and efficiently.

2- *Provide guaranteed QoS embedding:*

As illustrated in Fig. 2.1, VNRs are installed and administrated by VN operator with respect to a set of QoS constraints defined by SPs. These demands must be fulfilled by VNE algorithms performed by corresponding SPs. For example, a VN offering VoIP/outdoor broadcasting services requires high CPU, medium bandwidth, and low link propagation delays while a VN that provides P2P services must meet medium CPU, medium bandwidth, but none-delay bounds. This service can be handled by distributing equal workload across the SN [1].

3- *Provide survivable embedding:*

VNE resilience allows to integrate redundant resources in SN. Some primary substantial nodes/links or all of them can be set out a resilience strategy. In such, consistency factor is significantly important on network topology and network resources defined for failure situation. Moreover, transparent recover should be provided to users, which means that user should not notice that network already reserves to the

back-up resources, even with delay-sensitive applications. Hence, QoS requirements of the entities that may be selected for redundant resources should be attentively considered.

2.2.4 Problem Decomposition and Coordination

As mentioned in previous section, VNE problem can be divided into two sub-problems: VNoM and VLiM. Accordingly, solving VNE can be achieved by different approaches where each sub-problem is addressed in an isolated and independent manner. In this approach, VNoM is solved at first that provides the input for the next VLiM stage, which is also known as *uncoordinated VNE* deviation. An alternative approach called *coordinated VNE* may enhance the embedding performance by coordinating two different phases. The coordinated VNE itself can be dealt with in either two different and coordinated stages or just in a single stage. Embedding VNRs can be conducted across heterogeneous InPs, also known as InterInP coordination, aiming at splitting VNRs into various sub-requests and then looking for an appropriate InP to handle each of them. In contrast, all VNRs can be usually managed by a single InP in most of VNE approaches.

Uncoordinated VNE:

Uncoordination indicates that VNoM and VLiM are solved in two different stages. As such, output of VNoM becomes the input in VLiM stage. A representative example of this approach is introduced in [43] in which the main objective is to maximize the long-term revenue. VNoM in this approach is carried out by the Greedy algorithm which establishes a set of eligible substrate nodes that meets the resource demands, and then assigns one of them for each virtual node with respect to the amount of available resources. Depending on path-splittable restrictions of VN requests, VLiM can be handled by either *k-shortest path* algorithm [44] or Multicommodity Flow mechanism (MCF) [45]. If each virtual link must be embedded onto a single physical path, the former method is applied; however, if a virtual link allows to be mapped onto several substrate paths, VLiM is reduced to MCF problem that enables an optimal multi-path routing solution for each virtual link utilizing linear programming algorithms [46].

There are several pros and cons in terms of this mapping selection. VNE solutions without coordination between VNoM and VLiM might increase the embedding cost in the link mapping stage causing low acceptance ratio and, hence, low revenue in

long-term due to the neighboring virtual nodes being mapped far from each other in the substrate network topology. However, splittable-enabled mapping might even produce a better resource utilization in theory, but unsplittable methods seem more difficult to achieve optimum compared to splittable counterpart. Additionally, splittable-based solutions could generate abundant overheads for maintaining network state consistently [47] and result in the possible problem of out-of-order package delivery with extra latency that might not be acceptable for sensitive-delay applications. Furthermore, coordination solutions relied on optimal LP algorithms might exacerbate algorithmic implementations and time complexity.

Coordinated VNE:

coordinating node and link mapping is usually desirable. VNoM without considering its relation with link mapping might restrict the solution space and reduce the embedding performance. In general, VNE coordination can be performed in two stages or in a single stage.

- Coordinated VNE in two stages: An example of node and link coordination in two stages that we can discuss here is [48] in which its objective is to minimize the mapping cost. An augmented graph over the corresponding substrate network is constructed for a VN request. Over this graph, the proposed algorithm addresses VNoM by formulating a Mixed Integer Linear Programming (MIP). To reduce MIP complexity, the Integer Programming is relaxed to linear programming and then its achieved solution is rounded to solve VNoM in two different manners: deterministic and randomized rounding based VNE (corresponding to D-ViNE and R-ViNE algorithm respectively). Thereafter, VLiM is conducted as the same two link mapping solutions in [43]. Benefits of IP are that it is a mathematical modelling technique where its objective is to maximize or minimize a function subject with various constraints. IP is expected to achieve an optimal solution for offline VNE problems. Modelling IP formulations is very straightforward, which can be solved by an optimization solver. However, solving the IP formulations to achieve optimal solutions is computationally intractable, so IP is usually relaxed to linear programming (LP) that can be easy to solve in polynomial time. By introducing relaxation, we could scarify some degrees of optimality so the solutions are not optimal. In fact, there is no actual optimal solution in online VNE problems due to the fact that the information of future VNRs is

unknown.

- Coordinated VNE in one stage: this approach implies that virtual nodes and links are embedded simultaneously. After a pair of virtual nodes is embedded, the virtual links that connect these nodes will be also mapped. An example of this variant is [49]. Taking topological attributes into account, node ranking approach, inspired by PageRank algorithm deployed in Google's search engine, is used to measure topological impacts on a node. Node mapping considering topology attributes is expected to improve VN acceptance ratio and the link embedding efficiency. First, the proposed algorithm calculates that node ranks for both substrate and virtual nodes and then constructs a Breadth-First Search (BFS) tree of the corresponding VN request. VN embedding is performed by moving over the BFS tree and then embedding each virtual node to the highest-rank substrate node sequentially. At the same time, the algorithm also maps virtual links corresponding to already-mapped virtual nodes onto the shortest paths in the substrate network.

- InterInP coordination: previous variants just handle VN requests in a single-domain InP scenario, but in some cases, VN requests can be mapped on several substrate networks that are managed by different InPs. Each of them can handle parts of VN requests and connects them by external links among them. In each InP, these requests can be embedded by a VNE algorithm. In fact, there is a conflict of interest between SPs and InPs. SPs would like to have as many accepted VN requests as possible while minimizing the expenditure whereas InPs struggle to optimize the revenue by accepting VN requests. Samuel et al. in [50], for instance, addresses this issue by proposing a distributed protocol ensuring competitive mapping pricing for SPs while coordinating InPs.

2.3 Literature Review

There are numerous research endeavors in NV, where [1], [51], [40] and [39] have carried out comprehensive surveys in this fundamental topic. VNE is widely known as $\mathcal{NP}$-hard in nature, which is intractable to solve. In literature, VNE can be classified into three categories in accordance with their optimization strategies [1], but with the rise of machine learning techniques recently, we think that this category deserves its

own stand. By that statement, four methods for solving VNE problems include: (1) mathematical-based exact solutions, (2) heuristics, (3) metaheuristic algorithms and recently (4) machine learning approaches. Most research on VNE is concentrated on other methods except the exacts due to its computational complexity.

2.3.1 Exact Methods

Mathematical optimization algorithms embed each VNR using exact optimization strategy like Linear Programming (LP) [52], more specifically including Integer Linear Programming (ILP) or Mixed Integer Programming (MIP). After VNE problem formulated, open-source or proprietary software tools, known as solvers (e.g. GLPK, CPLEX, GUROBI), assist to solve the formulated ILP/MIP models. The exact algorithms are usually deployed for small instances of VNE problems and consequently used as benchmarks when compared with heuristics or metaheuristic algorithms due to their time complexity associated with high execution time problem [53]. For example: embedding a single VN of 10 nodes onto a substrate network with 40 nodes might take more than 30 minutes [39]. As a result, integer constraint relaxation method could be applied for dealing with the complexity issue so as to solve LP model in polynomial time. When a feasible node solution of VNoM is found, SP algorithm or MCF mechanism will be conducted to embed virtual links onto corresponding substrate paths. Unfortunately, the uncoordinated VN embedding isn't optimal in most cases [39]. In order to improve performance, a coordinated node-link formulation approach and column or path generation methods can be implemented to come towards optimal VNE.

Chowdhury et al. [12] proposed MIP model for VNE problem considering a coordinated node and link approach at the same time for VNoM. An initial version of [12] can be found in [48]. Due to high computational complexity of MIP model, the authors relaxed integer constraints to achieve a linear programming. Two rounding techniques including deterministic and random were exploited to select a distinct node embedding solution. If the LP solution is found feasible, SP or MCF algorithms were used to map virtual links onto substrate paths. These solutions have been widely accepted as an exact-like mechanism in VN mapping research. This research has been one of the most cited article in VNE, but its concentration was on node mapping, entrusting the shortest path method and MCF algorithm with load balancing support for VLiM.

Authors in [54] proposed VNE Node-Link Formulation (VNE-NLF), an ILP model, for mapping each VNR. Its objective was to minimize the VN embedding cost with different proposed cost functions while supporting load balancing. VNE-NLF basically took node capacity and link constraints into consideration. Although this model achieved favorable embedding results, it was not capable of practical networking environments due to high time complexity of pure ILP model. However, VNE-NLF could become a reference benchmark for comparing with other VNE solutions.

Mijumbi et al. in [55] deployed a column generation approach, namely PaGeViNE, to achieve feasible VN embedding solutions. PaGeViNE was formulated a pure MIP model as Path Generation approach for the VNE problem, then relaxed into LP to reduce its complexity. PaGeViNE procedures were similar to those in [12], but the corresponding dual MIP model here was derived to choose legitimate paths. The VNE solution that consumed less physical resources is preferable. This column generation mechanism considerably enhanced scalability. After the LP model was solved for VNoM, the shortest path method was selected for VLiM. In many scenarios, however, this algorithm usually fell into the local optimum.

Similar to [55], [56] and [57] also exploited path generation approach for addressing VNE problem. Hu et al. [56] formulated the VNR problem as a single stage path-based ILP model where each virtual link represented one commodity. The column generation method was then carried out to solve the relaxed mathematical model, namely LP-VNE. In fact, LP-VNE enabled path splitting for mapping virtual links, which was significantly simpler than unsplittable embedding in [55]. Jarray et al. proposed the column generation formulation for solving VNE problem in one stage applying an auctioning mechanism in which VNRs can be processed in batches. Thus, this approach could be recognized as an offline embedding [55].

Huang et al. in [58] extended [12] by considering the substrate node splitting and node collocation for solving VNE efficiently. The authors studied a realistic VNE scenario in which both virtual and substrate nodes were classified as either transit nodes responsible for replaying traffic, or stub nodes that originated/sunk traffic.

The authors in [59] presented the approximation algorithms for a performance-guaranteed VNE solution. They introduced a randomized rounding technique for LP model to maximize the total profit of embedded VNRs. However, the major problem with this method is that its predetermined strategy was used prior to embedding decisions, which made this model non-adaptive. Their algorithms focused on specific

cactus request graphs. In worse-case, using various demand-to-capacity ratios, the performance of their proposed algorithms seemed far away from optimal.

In [60], Cao et al. presented a candidate for reducing the computational complexity with less performance loss. The idea was that their algorithm constructed the substrate node and path subsets before embedding VNRs. Four node and link constraints were investigated to produce the more-applicable VNE solution. Unfortunately, their solution still had high time complexity that could not beat heuristic algorithms towards execution time.

Dietrich et al. [61] examined the VNE problem across multiple providers with limited information disclosure (LID). The framework, considered an information sharing scheme that formulated VN partitioning and segment embedding, included four steps: resource information disclosure, resource matching, VN partitioning and finally VN segment mapping. Each physical network embedded corresponding VN segments while meeting VNR requirements and then a MIP model similar to [48] was conducted to address VNE problem. The performance results reinforced the idea of an emerged business model for decoupling network operations and physical infrastructures.

2.3.2 Heuristic Algorithms

Different with exact methods, heuristic algorithms are not aimed at approaching an optimal solution for a VN. They try to balance the trade-off between finding a feasible VNE solution and maintaining low execution time. While reducing VNE execution time of heuristics is important to prevent possible delay in mapping a corresponding VNR, this target could face an issue of being stuck in local optimum. Several heuristic algorithms have been proposed to achieve better VNE solutions.

The authors in [62] dealt with the sequential one-by-one online embedding scenario in which VNRs arrived and departed dynamically. However, substrate network resources were assumed "infinite" and virtual node and link constraints (e.g. CPU, bandwidth) were not considered in this paper. Its objectives were to effectively map VNs onto SN while minimizing workloads on substrate nodes and links.

On the other hand, [43] presented the path splittable-enabled embedding of a single virtual link over multiple physical paths, relegating the link embedding problem to an MCF problem by considering a virtual link as a commodity. A simple heuristic algorithm based on a Greedy mechanism was proposed for the VNoM, then Dijkstra's

algorithm and MCF approaches were utilized to deal with the link mapping problem. The objectives were to increase the VNs acceptance ratio and to maximize the long-term average revenue. However, this solution exacerbated the resource fragmentation issue due to its inefficient approaches, and path migration even was mentioned in brief but not formulated.

Cheng et al. [49] proposed another node ranking method called RW-Max-Match, inspired by Google's PageRank algorithm, to address VNE problem. Network topology attributes including node degree and link strength were utilized in this approach. Link mapping algorithms were the same in [43]. VN acceptance ratio and resource utilization were proved to be partly enhanced.

Similar to [49], Gong et al. in [63] presented another node ranking algorithm for ranking all substrate nodes considering global resource information. Its link mapping was based on the shortest path algorithm. An extension of [63] formulated a global resource capacity (GRC) metric in [64] to quantify all substrate nodes for VN embedding potentials, namely GRC-VNE. The proposed algorithm, that referred highest GRC in SN and coordinated link embedding constraints in the VNoM stage, embedded VNs onto the corresponding SN. The shortest path method was also leveraged for VLiM stage like [65]. However, these papers did not study the possible effects of other topology attributes on VNE problem.

Additionally, Feng et al. in [66] investigated several topology attributes for VNE (e.g. degree, strength, farness centrality, closeness centrality, betweenness centrality, eigenvector centrality and Katz centrality) and accordingly proposed three topology-based node-ranking algorithms and three modified random walk algorithms. However, only three topological attributes were well exploited in corresponding three proposed algorithms [39]. Both virtual nodes and substrate nodes were ranked in decreasing order and the ones with highest node-ranking values were referred to node mapping. Dijkstra's algorithm was used for link embedding stage.

A node-ranking approach relied on node degree and node clustering coefficient information was proposed in [67]. Local resources of nodes and their neighborhood nodes were merely taken into account without the global network resourced and link topology attributes. This could be considered as another extension of [43], so its problem would be inefficient utilization of substrate resources in long run.

The authors in [68] suggested a novel node-ranking approach where topology attributes and global network resources were utilized in the node mapping stage

following a link mapping stage. This work considered both the local resources of the underlying node and its relationships with other nodes in the SN. The node-ranking method enhanced the possibility of choosing an appropriate node that satisfies node-link constraints while reducing network resource fragmentation. Similarly, the VNE-NTANRC approach in [69] extended [68] and [70] with five essential topology attributes accompanying the global network resources to achieve a set of ranked nodes for embedding.

Also inspired by the PageRank, [71] ranked substrate nodes using graph eigenspace alignment techniques by computing the eigenpairs of a graph. The authors formulated the VNE problem in the form of a graph-path distance matrix, and aligned the eigenspaces of virtual networks and substrate network to improve the VNE solution. They proposed a set of algorithms that involved several generalized distance metrics called DMEA-X and DMEA-2D, pursuing a trade-off between computation time and solution quality. These proposed algorithms strove to surpass the impressive embedding performance of DViNE-SP [12] while exceeding the fast run time of GAR-SP in [43]. Lischka et al. in [72] proposed the Subgraph Isomorphism Detection (SID) technique with a backtracking search approach for embedding virtual nodes and links in one-shot. Specific cases of SID problem could be resolved in polynomial time and the assignment of backtracking search might not exist. However, SID mechanism was incapable of promoting to online VNE since it had to know the SN and VNRs in advance. Furthermore, [73] proposed a topology-aware VNE approach to deal with possibly multiple substrate node failures. This work was different from other redundant VNE algorithms [74], [75], [76] and [77] in which only a single substrate failure was studied in a cost-effective manner. The paper [73] was aimed at improving long-term revenue while reducing the node-failure penalty. Consequently, the authors initially erected the Candidate Node Set for each virtual node. The one with highest probability among those in such node set was selected to replace the already-mapped substrate node when the corresponding substrate node failed. A recoverability-based VNE algorithm was proposed to solve VNs in the first stage, that was also another objective in the formulated objective function aimed at mapping virtual nodes onto the substrate nodes with higher recoverability. A profit-driven node mechanism was deployed to re-embed the node mapping when substrate node failures happened.

One of the first research considering redundant resources in VNE was [78], that formulated a survivable network embedding (SVNE) problem based on the reference

model in [65], considering that a SN was able to partially fail. It addressed a problem of a single link failure while explicitly neglecting node failures since any node failures in VNE were dependent on adjacent link failures. Due to the $\mathcal{NP}$-hard nature of the MIP formulation of SVNE, the authors proposed three heuristic algorithms where VNoM and VLiM were solved separately with path splitting support. Their objectives were to maximize the embedding revenues and minimize the consequences caused by a link failure. As the probability of multiple link failures might be low, fast rerouting algorithms were investigated to mitigate the failure of a single link. However, the major drawback of such approaches was possibly the inefficiency of allocated network resources.

Khan et al. in [79] extended [80] advised another survivability model in VNE, namely SiMPLE, addressing a single physical link failure with path splitting support. The authors' assumptions were based on the facts that physical node failures could be modeled as multiple physical link failures [81]; and link failures are usually more common than its counterpart - node failures [82] [83]. Besides, Markopoulou et al. in [83] indicated that at most 5% of the physical link failures existed in the process of another substrate link failure. As a result, the authors investigated a single link failure in VNE. To solve large network instances, two heuristics of ILP of SiMPLE model were proposed including proactive and reactive solutions. The formal mapped a VN and made a resource backups at the same time while the latter was adapted the substrate link failure when it occurred. Due to enabled path splitting, resource backup was conducted on $1/k$ virtual link bandwidth request where k represented the average number of splits on embedding a virtual link. This simple idea would help save more network resources for accepting more arrived VNRs.

Likewise, [84] extended [78] by introducing an alternative model for the Survival VNE problem, namely Connectivity-aware VNE, which mapped a VN onto a SN, ensuring VN connectivity under a limited number of substrate link failures. Its goals were to reduce the requirements for backup resources compared to conventional SVNE models, while minimizing embedding costs.

In addition, [85] proposed two Location-Constraint VNE (LC-VNE) algorithms based on a compatibility graph (CG) to solve the integrated node and link mapping problems. The authors transformed the LC-VNE problem into a minimum-cost maximum clique (MCMC) problem. This VNE approach still encountered the problem of resource fragmentation and extravagant resource utilization and locations of different

SNs were omitted.

As a conflict between SPs and InPs in VNE where each InP tended to exploit their substrate resources by accepting as more VNRs as possible to increase more profits while each SP aimed to minimize the resource consumption of the embedded VN requests, the VNE problem across multiple administrative domains handled by different InPs was studied in [50]. The policy-based inter-domain VNE algorithm, namely PolyViNE, mapping VNs in a decentralized manner was proposed. The proposal introduced a distributed protocol that coordinately handled VNE process across multiple engaged InPs while establishing competitive prices for SPs. Accordingly, PolyViNE investigated a hierarchical address system (COST) and a location dissemination protocol (LAP) that allowed InPs to inform their embedding availability with corresponding costs during matching process and then SP selected the one with lowest cost. InPs could jointly provide an embedding service to a VN. However, each VN was allocated to a single SN and splitting support was not raised, so this proposal could be considered a special case of distributed VNE problem [39].

In addition, Mano et al. [86] presented a search algorithm that was based on secure multi-party computation (MPC) to embed VNs across multiple SNs without leaking any private information of SNs efficiently. First, each VN was divided into pieces and then their price order was calculated, making up MPC sorting phase. The second step was to choose a preferred set of pieces to handle the corresponding VN. Eventually, SNs specified the minimal number of pieces based on the Pareto optimality set to reduce the search space.

The work in [87] extended [88] to investigate the parallelizable VNE, namely EPVNE, which was aimed at minimizing the embedding cost while increasing the acceptance ratio. The physical network was assumed to support parallel computation and permit a virtual node to be possibly mapped onto multiple substrate nodes. EPVNE simply embedded each virtual node onto a single substrate node; but if the network could not adapt to the virtual node CPU resource, it was able to be mapped onto multiple physical nodes. Following node stage, the link mapping was carried out by the shortest path mechanism. Even EPVNE was a very straightforward heuristic algorithm, but it outperformed two more complicated solutions including LazyP and ProactiveP in [88].

Dehury et al. [89] proposed a dynamic heuristic VNE algorithm (DYVINE) that was based on self-developed fitness values. Their algorithm solved the problem of users

being able to change VN resource requirements and the structure of a VN dynamically after it had been embedded. DYVINE was composed of migration and re-embedding processes. Although resource utilization of a substrate node was considered in the node mapping stage, connection states among nodes were neglected. This deficiency might result in an inefficient link embedding scheme that can meet VNE embedding constraints.

Most current heuristics concentrate on ranking approaches that usually exploit predefined topology attributes and network resource states to decrease searching space. Virtual nodes are then allocated one by one utilizing these ranking techniques. When all the substrate nodes have been embedded, the shortest path algorithm or MCF approach is then used to interconnect them. Although heuristic approaches can somehow guarantee the embedding speed, they have to strive for performance efficiency. Due to fast characteristic, heuristics are widely and highly applied in solving online VNE problems.

2.3.3 Metaheuristic Algorithms

In contrast, metaheuristic methods attempt to compromise optimal mapping solutions with shorter execution time. They can achieve near-optimal solutions by improving the quality of candidate solutions through quality measurement and evolution strategies. However, this research trend is not mainstream in VNE field [39]. We hope that this research proposal can revive metaheuristics back to the mainstream thanks to their appealing performance in this research with regard to the distributed and parallel operation scheme.

Genetic Algorithm has been widely applied in the Artificial Intelligence field. Its application to VN embedding problems was first investigated in [90] and [91]. Mi et al. in [90] proposed node ranking methods that relied upon two GA algorithms with multiple topology attributes (CB-GA and RW-GA) to rank a given node. CB-GA was based on the cost and residual bandwidth of all outgoing physical links of the corresponding node in the substrate network. In contrast, RW-GA utilized the Markov Random Walk model to determine the substrate node hosting a virtual node based on its remaining resources and topological attributes. Evaluation results showed that these proposed GA algorithms performed better than PSO-based VNE algorithms.

Chang et al. [91] ultimately compared the performance of several metaheuristic

algorithms, including Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO) and GA. The authors introduced two VNE algorithms based on ACO metaheuristic and GA algorithms. However, they assumed that the performance environment was restricted and the proposed algorithms are focused only on the node mapping stage.

Zhou et al. in [92] embedded multiple VNRs at the same time under batch arrivals, where the substrate resources were assumed not to be sufficient to meet all VNRs in the corresponding batch. GA algorithm was advised the network to decide which VNRs to be embedded together. The resource-ware manner was considered to maximize the long-term revenue of SP. VNoM and VLiM processes were conducted separately in which MCF method was deployed in the latter stage. Although the proposed solution was better than the compared algorithms in terms of acceptance ratio and long-term revenue, the embedding was not indeed efficient.

Likewise, [93] studied the efficiency of GA algorithms in handling multiple InP domains with a VN embedding model, while the authors [94] successfully revised a conventional GA algorithm by reordering the mutation stage right after the initial population to regenerate better quality offspring.

Targeting the coordinated VNE problem, an interesting paper that proposed Mod-Maxmatch algorithm for VNoM and GA algorithm with path splitting support for VLiM presented in [95], addressing the coordinated VNE problem. Mod-Maxmatch algorithm considered the global link resources for VNoM while GA determined the path and splitting ratio to allocate a virtual link. The path among multiple generated ones selected by GA was produced by the shortest path algorithm. GA was targeted to tackle the phenomena of link congestion and enable dynamic link mitigation. However, only average link cost was evaluated on a small scale of substrate network (25 nodes, 40 links) and the time complexity was high since the breadth-first search (BFS) method and typical GA (with Dijkstra algorithm in initial link process) were used in VNoM and VLiM respectively. Furthermore, no information about other important performance metrics (e.g. revenue, network utilization, acceptance ratio) were shown, so it would be hard to evaluate the efficiency of this proposal.

The problem of CPU capacity of Intermediate Substrate Nodes (INSs) was raised in [96] to forward traffic on virtual links at different required rates. This obstacle had been mostly overlooked in the literature. The authors concluded that there was an impact on VNE mapping performance when considering the CPU utilization of INSs

with the desired forwarding rates of virtual links. Several metaheuristics including GA, PSO and HS algorithms were evaluated in this paper. In practise, this practically depended on the ISNs' capacity as well as on the actual size of the substrate network.

Boyang et al. in [97] introduced a hybrid adaptive GA algorithm for VNE problem. The authors proposed a new crossover operator to increase the convergence speed while a simulated annealing was adopted to replace typical mutation operator of GA algorithm in order to avoid possible local optima. Remapping strategy enabled to remove unreasonable mated individual from GA procedures. Eventually, the shortest path mechanism was applied to VLiM stage.

In [98], Zhang et al. proposed a genetic correlation multi-domain VNE algorithm (GCMD-VNE) that was based on a GA algorithm where the selection operation followed a stochastic method and the crossover was not conducted if two parents were the same. The proposed GA-based algorithm was compared with PSO-based approach and two heuristic algorithms in terms of average embedding cost and execution time only. Moreover, other crucial constraints (e.g., propagation delay, node locations) were omitted. Even GA performed better than PSO and other heuristics, but it had slow convergence and faced high time complexity.

In terms of ACO solutions, Fajjari et al. in [99] proposed a typical ACO algorithm for VNE problem in aim at minimizing the allocated substrate resources while maximizing the long-term revenue of SP. As an extension of [99], [100] took both substrate node capacity and the distance between virtual and substrate nodes (link distance message) into consideration in order to reduce the VN mapping cost while improve the revenue in VNoM phase. For link mapping phase, [100] applied the shortest path algorithm for finding a feasible link solution.

The authors in [101] built a preferable embedding sequence of virtual nodes according to their sorted link resources. ACO was proposed to map the virtual nodes onto substrate nodes based on the ranked sequence, then the virtual links could be embedded using the shortest path method for the corresponding embedded nodes.

Zhu in [102] proposed a modified ACO algorithm relied on graph decomposition. A pre-computing process based on ant random walking to accelerate the characteristics of ring topology and its results were utilized to guide VN decomposition and embedding stage regarding the ring structure. Topology of VNs was disintegrated into ring and tree structures. Different mapping algorithms were advised for each specific structure. By such idea, VLiM could be conducted simultaneously by a point-disjoint path search

mechanism. However, such decomposition solution would not efficiently adapt to heterogeneous VNRs in practice.

On the other hand, Song et al. in [103] presented a distributed framework for solving distributed VNE problem, that was based on historical archives (HA) and set-based metaheuristic PSO algorithm, namely HA-VNE-PSO. Archives recorded the embedding history and then utilized this information for embedding incoming VNRs while PSO was considered as an optimizer. However, the proposal encountered high time complexity as well as slow convergence and could not address VNE problem under SN failure.

[104] extended [103] by proposing a dual-heuristic PSO solution for solving VNE problem in one-stage. DH-PSO, a proposed solution, updated particles sequentially and then dual heuristic strategy was used to improve the VNE solutions. The first heuristic constructed a candidate set while the the second sought for the best solution from these candidates. The authors considered network resources and link feasibility to erect the potential VNE solutions. Another approach that also borrowed the idea of working animals (e.g. bees) was proposed in [105] for VNE problem, namely artificial bee colony (ABC). ABC idea came from the group of bees gathering honey where bees were classified into three groups: employed, onlooker and scout bees. Employed and on-looked bees were responsible for updating better solutions through iterations while the scout bees were expected to prevent local optima. The authors proposed two embedding strategies including preferential exchange to increase the convergence speed and the pheromone and sensitivity model to prevent local optima. The proposed algorithm competed PSO counterpart in all evaluation metrics. Similar to [105], ABC technique was studied in [106] to deal with dynamic VNE problem across multiple ISPs. A penalty function and a scaling factor based search were designed to improve the solution quality and the global convergence.

Regarding PSO solutions, Wang et al. in [107] formulated a discrete PSO (DPSO) model for VNE problem and proposed a particle re-construction mechanism with probability selection to achieve an efficient VNE solution. Through simulation, the proposal slightly improved convergence performance and embedding quality.

Similarly, [108] formulated a multi-objective VN re-configuration model to improve substrate resource utilization and reduce the overload problems on substrate nodes and links. This work also deployed DPSO to solve the re-embedding problem on the running VNs. However, this solution would cause some possible issues such as

overhead, data integrity, security. The embedding gain from such proposal was not really high.

DPSO was also utilized in [109] to solve the node consolidation (called "repeatable embedding" in the paper) in the SN, allowing multiple virtual nodes in the same VN to be embedded onto the same substrate node if it had sufficient resource capacity.

Zhang et al. in [110] also leveraged discrete niche PSO technique to address VNE problem with two objectives: maximizing revenue while minimizing energy cost. The proposed solution, called MO-PSO, outperformed the compared algorithms in terms of corresponding energy consumption, revenue and active nodes. Furthermore, the authors compared the running time between algorithms and they proved that their PSO solutions could certainly compete D-ViNE-SP in [12] in this performance metric.

In [111], the authors investigated PSO algorithm, namely VNP-PSO, to enhance the VN partitioning across multiple physical domains while keeping the mapping cost minimized. VNP-PSO was able to approach the embedding performance of an exact limited information disclosure method.

The PSO algorithm in [112] addressed the VNE problem across multiple domains relied on multi-controller SDN paradigm, aimed at minimizing mapping costs. In this paper, global controller collected the estimated information from local controllers (e.g. mapping overhead, candidate nodes) in their local domains and then applied PSO algorithm to figure out a division solution for each VNR embedding. However, the communication overheads between controllers were not quantified and delay might be worse when the network became busy.

Rubio et al. [113] proposed an enhanced VNE algorithm based on metaheuristic approaches including GA, PSO, Harmony Search (HS), and the Firefly Algorithm (FA). All of the algorithms chose certain VNE solutions as potential candidates for the corresponding VNR, regardless of their feasibility. They could be combined to produce other VNE solutions during the metaheuristic search procedures. The candidate that met the VN's constraints had a greater chance of being selected. Through their evaluation performance, however, it appeared that the quality of their embedding solutions was not consistently maintained in several cases.

Compared to heuristics, metaheuristic approaches do not have an advantage on speed but they can achieve the efficiency goals that are significantly desired by network providers to exploit the network resources efficiently. Moreover, almost metaheuristics can be easily deployed in a parallel manner, which opens a chance for overcoming their

slow mapping speed in nature. The aforementioned metaheuristics mostly concentrated on the VNE node mapping stage leaving VLiM for the shortest path methods or MCF like heuristic algorithms. To date, no sophisticated VNE algorithm has been sufficiently studied for VLiM. Recently, our previous papers have revisited unsplittable VLiM [36, 37] and coordinated VNE problems [38]. The differences between these papers and our contributions in this dissertation are that [36–38] allows possible modifications on parental genes which means that new genes can be generated after crossover and mutation operations.

2.3.4 Machine Learning Approaches

AI and ML have been recently hottest topics as these technologies increasingly get into every scientific fields when ML applications enable automation in dealing with complex problems in diverse domains. In fact, their applications have gained promising improvements arising in network operation and management towards performance efficiency [114]. In terms of VNE problems, it is laborious, burdensome and very expensive to collect sufficient labelled data, so supervised learning algorithms are unlikely a preferable choice in most of research papers in this field. Thus, we focus on reinforcement learning algorithms to solve this problem in this section.

In 2014, Mijumbi et al. [26] proposed a distributed reinforcement learning to dynamically allocate substrate resources to requested VNs. In this work, information of allocated resources, unused virtual/substrate resources under percentage was taken into account to make up a lookup table with different states driven possible actions of agents in learning algorithm. This strategy was aimed at ensuring that VNRs were not rejected if network resources reserved to already-mapped VNs were still available. The VNE solution based on the look-up table was replaced by a Feed Forward Neural Network in [27]. These papers handled resource allocation problem using a tabular Q-learning algorithm and a fully-connected NN where the percentage of available resources of each substrate node as input features. Unfortunately, the authors limited the state-action space, causing sub-optimal solutions.

In [32], Recurrent Neural Networks (RNNs) were deployed for an admission control to enhance embedding performance by preventing spending time on infeasible VNRs or not embedding in acceptable time, which could reduce time complexity. Topological and resource features were extracted from substrate network and VNRs composing of network presentation models. These attributes were utilized as inputs in RNN

model for predicting solvable VNRs given historical experience from previous records. However, the main goal was to reduce system runtime through an improved admission control using RNN, that was also expected to enhance the embedding performance. The feature representations were still simple considering CPU and bandwidth resources in this model.

NeuroViNE [115] leveraged the Hopfield network, a special form of RNNs, in order to pre-process VNRs by getting the extractions of their corresponding subgraphs, then selecting a subset of substrate nodes to embed VNRs. NeuroViNE constructed a node and path ranking based on high remaining resources and the shortest paths between nodes. Hopfield could generate an increasing number of pre-selected nodes, that decreases the efficiency of NeuroViNE. Moreover, reducing searching space for runtime operation could affect the embedding efficiency.

Haeri and Trajkovic [116] exploited the Markov Decision Process (MDP) to solve the VNE node mapping and then employed the Monte-Carlo Tree Search (MCTS) method as the action policies to maximize revenue and minimize embedding costs. The authors proposed two MaVEn algorithms where a simple breath-first search algorithm (MaVEn-S) and MCF method (MaVEn-M) were used to solve the VLiM problem. Although their solutions performed better than the compared algorithms, they encountered the problem of high time consumption since MCTS has to run the whole procedure for every mapping decision. Moreover, the reward function that guides MCTS algorithms was quite simple, which was insufficient to represent the complicated environment of VNE problem. MCTS also faces an obvious problem of scalability when the state-action space becomes large. Only MaVEn-S could be comparable to D-ViNE and R-ViNE algorithms in average processing time while MaVEn-M consumed a lot of time to process a VNR, so it is not suitable for online VNE problem.

Additionally, Yao et al. [19] proposed 1-dimensional convolution neural network (CNN) to extract four features of each substrate node utilized as input to the policy network, and trained NN by minimizing cross-entropy loss function that then exploited the selected action as handcrafted label. For simplification, the policy network was kept as simple as possible with 4 layers and one filter. Substrate nodes yielding higher probability after filtering were more likely selected. The feature matrix only considered a spatial information of SN for extracting characteristic features. To select the appropriate nodes, a simple filter based on CPU capacity was deployed. Similar

to [116], the simple reward function was relatively relied on revenue to cost ratio to drive the agent learning process. For simplified training, the authors also assumed that all VNRs followed an invariable distribution, which fell into sub-optimal solutions. Another extension of [19] was presented in [117] that proposed a RL model using a dynamic attribute matrix representation (RDAM) to deal with the VNE problem. This model considered network topology attributes and network structures under the matrix form. Due to the extraction of node attributes, the RL agent was able to decipher the SN and then learn embedding rules. The RL algorithm determined the relationship between the SN presentation and VNRs. However, RDAM algorithm focused only on node mapping and applied a Breadth-First Search (BFS) method for the link mapping stage. It also faced the same issues as its predecessor [19].

He et al. in [29] formulated the VNE problem as MDP. The authors deployed an improved Q-Learning algorithm and a curiosity-driven mechanism considering the energy factor and acceptance ratio as the major optimization objectives to solve the VNoM stage. The proposed algorithms came up with a trade-off solution that dealt with the Exploration-Exploitation dilemma and most likely tended to fall into local optima. As the most popular choice in a large number of VNE papers, the shortest path algorithm was used for VLiM stage. Unfortunately, there were no details in training and testing phases provided. By overlooking virtual links or link attributes as well as defining a simple representation for the environment, VNE solutions were far more optimal.

Similar to [29], the authors in [30] also modeled VNE problem as MDP, and then proposed a NN to approximate the value function of VNE states. To solve the model, a kind of RL techniques, namely Temporal Difference Learning (VNE-TD) was adopted for online VNs embedding. A probabilistic method generated node mapping candidates and then the TD-Learning evaluated the potential for each candidate. The one with highest values was selected for embedding the given VNR. Following VNoM, the shortest path method was utilized for VLiM. In fact, all possible states were used as the input for a NN as the same with [29], which increased the computational complexity. Additionally the authors exploited only SN to represent the environment, which was not sufficient to reflect the complex VNE problem.

In [20], RNN was used to model the continuity of the node mapping process in time series formulated as a classic seq2seq, aimed at exploiting the historical states of the SN. A continuous-decision VNE scheme based on RL relied on policy-gradient

mechanism to update the RNN parameters with respect to average long-term revenue-to-cost ratio, namely CDRL. The authors defined the environment as the same [19] but the distance factor was removed; again, it did not take VNR's demands as well as link attributes into account, that might cause the proposed algorithm less efficient. Besides, VNoM and VLiM stages were conducted separately where VLiM was relied on BFS algorithm.

Andreoletti et al. in [21] suggested a privacy-preserving RL algorithm for solving VNE problem across multi-domain infrastructure under the Shamir Secret Sharing (SSS) scheme where users and ISPs ciphered secretly. The authors compared the proposed solution with two rivals including Limited Information Disclosure (LID) and Full information Disclosure (FID) in terms of embedding cost only. Due to very high overhead, they performed a trade-off evaluation between the number of RL operations and the mapping costs without compromising the privacy. The original objective was to focus on privacy, embedding VNs was relied on two heuristic VNE algorithms that were based on a relaxed integer programming formulation, which could encounter the time complexity problem.

One of the first papers that applied Deep Reinforcement Learning (DRL) technique for VNE problem was presented in [31]. In this paper, the authors extracted features of both virtual and substrate networks, and then encoded them into two-dimensional images which were then processed by a convolutional deep NN. They limited a number of actions to boost up the learning process. The reward function was aimed at minimizing the blocking ratio, which was not comprehensive enough to address VNE problem. Multiple virtual nodes were able to be embedded onto a single substrate node in their work, which is a close form of virtual network scaling in NFV field. Furthermore, a generalized version of CNNs, known as Graph Convolutional Networks (GCNs) and DRL were proposed for VNE problem in [33]. Both papers coordinated VNoM and VLiM in which the shortest path mechanism was used to embed virtual links. However, these papers had to struggle with the problem of scalability so that they only applied the solutions on a small scale. VNs were defined as static in [31], that fixed to 9 nodes for each VN while the SN had 25 substrate nodes. The VNE solution was neither generalized nor scalable.

To automatically detect and provide a proper embedding solution to dynamic environment, a combination between CNNs and paralleled DRL was advised in [24]. Its original version was [23]. The authors proposed a parallel DRL training framework

with multi-objective reward function where DRL performed as a feature extractor in random graph topology. A hybrid mechanism that applied either the shortest path or edge-disjoint path set was used for coordinately finding virtual link solution during node mapping stage. The learning agent decomposed the embedding process of a VNR into a sequential mapping of virtual nodes and their corresponding virtual links, but if they were high correlation (e.g., tree topology) and one or more virtual links failed to be mapped, the mapping process could be more complicated and consumed too much time. For simplification, no link features and only CPU and bandwidth attributes were considered in the state presentation, which did not fully reflect the complex VNE environment. In this paper, environments were limited to few stationary scenarios, which did not fully reflect the extremely dynamics of online VNE environments. Furthermore, training agent was implemented in parallel with a large number of worker agents being trained in several hours, a lot of episodes and millions of VNRs for training. These costs were very expensive for online VNE problem.

Several RL-based approaches have been proposed to solve VNE problem. To achieve efficient mapping solutions, all possible network states associated with both SNs and VNs (e.g., substrate resources, virtual resource demands, already-mappings, pending mapping) have to include in order to represent the stationary environment. Unfortunately, almost current RL-based solutions merely exploited few network attributes of SNs whereas very few papers utilized a little information of SNs and VNs for training. Additionally, VNoM and VLiM both should be considered coordinately under a graph mapping in RL approaches to achieve the optimal goal. For implementation simplification, most of the papers, however, handled VNoM and VLiM separately, and usually focused on VNoM while entrusting VLiM to the shortest path methods. These RL-based approaches were indeed relied upon parametric models, which were highly dependent on stationary environments. Each model was well trained for a specific model, and might work well on a specific environment. In online dynamic environments, there would have a large number of models to be trained. Online training and embedding methods (e.g., [116]) faced significant delay to achieve an efficient VNE solution which is not suitable for online mapping whereas offline pre-trained approaches commonly struggled with the curse of dimensionality which was very difficult to solve. To handle this problem, those papers reduced the complexity of dynamic VNE environments to a few limited stationary scenarios, which made them in-adaptive and unresponsive to online VNE applications in reality. A comprehensive

summary of the popular VNE approaches is provided in Table 2.1.

Reference	Types	Resource constraints	Coordination	Main contributions	Shortcomings
Chowdhury et al. [12, 48]	Exact-like	CPU, link bandwidth	Coordinated	First MIP model proposed for VNE problems considering a coordinated node and link approach for VNoM	Both relaxed the integer constraints and turned MIP to LP model with two rounding techniques.
Melo et al. [54]	Exact	CPU, link bandwidth and node locations	Coordinated	Node-Link Formulation (VNE-NLF), a pure ILP model, was proposed for mapping VNRs.	High time complexity due to pure ILP model.
Mijumbi et al. [55, 56]	Exact	CPU, link bandwidth and node locations	Uncoordinated	A pure MIP modeled as Path Generation approach for VNE problems.	MIP relaxed to LP to reduce the complexity, scarifying optimality.
Jarray et al. [57]	Exact	CPU, link bandwidth	Coordinated	ILP modeled as an auction-based Join Node and Link Embedding using Path Generation with branch-and-bound technique and rounding-off approaches.	Batches of VNRs processed as an offline embedding, nonscalability
Hu et al. [56]	Exact	CPU, link bandwidth	Coordinated	A path-based MIP model for the VNE problem using Path Generation with branch-and-bound technique.	The shortest path algorithm runs several times to generate path commodities just for processing a VNR, causing high time complexity.
Huang et al. in [58]	Exact	CPU, link bandwidth and node locations	Uncoordinated	LP formulations considering node splitting and node collocation for the first time.	High time complexity due to the pure LP model.
Rost et al. [59]	Exact	CPU, link bandwidth	Coordinated	Decomposable LP formulations with randomized rounding techniques focusing on specific cactus request graphs	Non-adaptive model, offline VNE.
Cao et al. [60]	Exact	CPU, link bandwidth, node locations, link propagation delay	Coordinated	A pure ILP approach with less computational complexity with candidate-assisted constructions	High time complexity due to pure ILP model.
Dietrich et al. [61]	Exact	CPU, link bandwidth, node locations, prices	Uncoordinated	Multi-provider ILP VNE formulations transformed to LP using relaxation and rounding techniques with limited information disclosure.	Sub-optimal solutions and scalability problems.
Zhu et al. [62]	Heuristic	Only VN topology	Uncoordinated	VN assignment with/without reconfiguration approaches.	Infinite substrate resource assumptions, none-virtual resource constraints were considered.
Yu et al. [43]	Heuristic	CPU, link bandwidth	Uncoordinated	Substrate path splitting and migration	Resource fragmentation problem.
Cheng et al. [49]	Heuristic	CPU, link bandwidth	Uncoordinated	Topology-aware node ranking approach based on Markov random walk model.	Local optima, inefficient resource utilization.
Gong et al. in [63, 64]	Heuristic	CPU, link bandwidth, node locations	Uncoordinated	Another node ranking method considering global resource information.	Limited network attributes considered, local optima.
Feng et al. in [66]	Heuristic	CPU, link bandwidth, node locations	Uncoordinated	Three topology-based node ranking algorithms and three modified random walk methods considering several topology attributes.	Re-ranking SN and each given VNR, inefficient resource utilization.
Zhang et al. [39]	Heuristic	CPU, link bandwidth	Uncoordinated	A node ranking approach relied on node degree and node clustering coefficient information.	Inefficient utilization of substrate resources.

Cao et al. [68–70]	Heuristic	CPU, link bandwidth	Uncoordinated	Five network topology attributes and global network resources altogether considered for node ranking.	VNRs processed in batches and in-adaptive node ranking.
Zhao et al. [71]	Heuristic	CPU, link bandwidth	Uncoordinated	A node ranking method using graph eigenspace alignment techniques by computing the eigenpairs of a graph	Trade-off between high computation time and performance efficiency.
Lischka et al. in [72]	Heuristic	CPU, link bandwidth	Coordinated	Subgraph Isomorphism Detection technique with a backtracking search approach.	In-applicability to online VNRs.
Authors in [73–80, 83, 84]	Heuristic	CPU, link bandwidth, node locations	Uncoordinated	A topology-aware survival VNE approach to handling single/multiple node failures with failover VN remapping algorithms.	Inefficient resource utilization.
Gong et al. [85]	Heuristic	CPU, link bandwidth, node locations	Coordinated	Location-constrained VNE formulation transformed into a minimum-cost maximum clique problem leveraging the graph bisection problem.	Resource fragmentation and extravagant resource utilization.
Chowdhury et al. [50]	Heuristic	CPU, link bandwidth, node locations	Uncoordinated	A policy-based inter-domain VNE paradigm embedding end-to-end VNs in a decentralized manner with a distributed protocol.	A special case of distributed VNE problems where a VNR was only allocated to only a single SN.
Mano et al. in [86]	Heuristic	CPU, link bandwidth, prices	Uncoordinated	A search algorithm for Inter-InP VNE problem using minimal MPC operations without sharing the InPs' private information to minimize VN prices.	Sub-optimal and scalability issues.
Authors in [87, 88]	Heuristic	CPU, link bandwidth	Uncoordinated	A study of parallelizable VNE problems with an efficient parallelizable VNE algorithm.	Sub-optimal solutions.
Dehury et al. [89]	Heuristic	CPU, link bandwidth, link propagation delay	Uncoordinated	A fitness-based dynamic VNE algorithm with mitigation and re-embedding processes, allowing dynamic VNRs where their structure and resource demands can be changed during execution time.	Inefficient link mapping solutions by neglecting connection states.
Mi and Chang in [90, 91]	Metaheuristic	CPU, link bandwidth, node locations and link propagation delay	Uncoordinated	Metaheuristics in VNE problems with two GA-based and two ACO-based algorithms for VNoM.	Low mapping efficiency.
Zhou et al. in [92]	Metaheuristic	CPU, link bandwidth, node locations	Uncoordinated	Multiple VNRs embedding based on GA algorithm under batch arrival for VNoM.	Low mapping efficiency and delay violation for delay-sensitive requests.
Pathak et al. [93]	Metaheuristic	CPU, link bandwidth	Uncoordinated	Multi-InP VNE problem enabling to embed multiple VNRs, based on GA algorithm	High time complexity and complex implementations.
Aguilar-Fuster et al. [96]	Metaheuristic	CPU, link bandwidth	Uncoordinated	An impact of CPU capacity of Intermediate Substrate Nodes in VLiM considered forwarding rate requirements.	Low mapping efficiency.
Zhang et al. [94]	Metaheuristic	CPU, link bandwidth	Uncoordinated	Modified GA-based approach for VNE problems reordering the mutation stage right after the initial population to regenerate better quality offspring for VNoM stage.	A trade-off between time complexity and solution quality, very few performance evaluations.

Huang et al. [95]	Metaheuristic	CPU, link bandwidth	Coordinated	A coordinated VNE based on a Mod-MaxMatch approach considering global link resources for VNoM and splittable GA-based algorithm for VLiM.	Scalability issues and few performance evaluations.
Boyang et al. in [97]	Metaheuristic	CPU, link bandwidth, node locations	Uncoordinated	A hybrid adaptive GA-based approach with an adaptive crossover, stimulated annealing-based mutation and remapping strategy	High time complexity.
Zhang et al. [98]	Metaheuristic	CPU, link bandwidth	Uncoordinated	A genetic correlation multi-domain GA-based algorithm.	High time complexity, local optima and very few performance evaluations.
Authors in [99–101]	Metaheuristic	CPU, memory, link bandwidth, node locations	Uncoordinated	ACO-based approaches considering the distance between virtual and substrate nodes or the ranked sequence of link resources.	Slow convergence.
Zhu in [102]	Metaheuristic	CPU, link bandwidth	Coordinated	A modified ACO algorithm leveraging graph decomposition with ant random walking for pre-computing process.	Limited topology structures, location omitted and in-adaptive approach.
Song et al. in [103]	Metaheuristic	CPU, link bandwidth	Uncoordinated	A distributed VNE framework based on historical archives for embedding history recording and set-based PSO algorithm as an optimizer.	High time complexity and slow convergence.
Song et al. [104]	Metaheuristic	CPU, link bandwidth	Coordinated	A dual-heuristic PSO solution for solving VNE problem in one-stage	Limited performance evaluations.
Li et al. [105]	Metaheuristic	CPU, link bandwidth	Uncoordinated	Artificial bee colony (ABC) approach with two embedding strategies for solving VNE problem.	Local optima and inefficient utilization.
Pathak et al. [106]	Metaheuristic	CPU, link bandwidth	Uncoordinated	Multi-InP VNE problem based on ABC approach for dynamic VNRs.	High time complexity and slow convergence.
Wang et al. in [107]	Metaheuristic	CPU, link bandwidth	Uncoordinated	Discrete PSO VNE approach with particle reconstruction	Only time execution and convergence compared, high time complexity.
Yuan et al. [108], [109]	Metaheuristic	CPU, link bandwidth	Uncoordinated	A multi-objective VN reconfiguration model based on discrete PSO algorithm.	Large overhead, data integrity, security issues and inefficient solution.
Yuan et al. [109]	Metaheuristic	CPU, link bandwidth	Uncoordinated	A discrete PSO algorithm to solve node consolidation in VNE, allowing multiple virtual nodes in the same VN to be embedded onto the same substrate node.	Limited performance evaluations.
Zhang et al. in [110]	Metaheuristic	CPU, link bandwidth, node locations, energy cost	Uncoordinated	A discrete niche particle swarm optimization technique to design a PSO algorithm for solving VNE problems.	Lack of many important evaluations (e.g., acceptance ratio, resource utilization)
Guo et al. [111]	Metaheuristic	CPU, link bandwidth	Coordinated	A multi-domain VN partitioning approach based on PSO algorithm to increase the efficiency of VN partitioning with LID.	Limited performance evaluations.
Ni et al. [112]	Metaheuristic	CPU, link bandwidth	Coordinated	A PSO-based multi-domain VNE approach on multi-controller SDN paradigm	High communication overheads and only cost evaluations conducted.

Rubio-Loyola et al. [113]	Metaheuristic	CPU, link bandwidth	Uncoordinated	Enhanced metaheuristic-based approaches based on GA, PSO, HS and FA with a stochastic ranking algorithm based on a Bubble-Sort-Like Procedure.	Low performance efficiency and high time complexity.
Lu et al. [36, 37]	Metaheuristic	CPU, link bandwidth, node locations	Uncoordinated	Distributed and parallel GA-based approaches with novel crossover and mutation operations for VLiM in VNE.	Sub-optimal solutions.
Lu et al. [38]	Metaheuristic	CPU, link bandwidth, node locations	Coordinated	A coordinated one-stage VNE approach based on a distributed and parallel operation scheme.	Complex implementations.
Mijumbi et al. [26, 27]	RL	CPU, link bandwidth, node locations, link delay	Coordinated	A distributed multi-agent reinforcement learning for dynamic resource allocation in VNE.	Large communication overheads, nonscalability, stationary environments.
Blenk et al. [32]	ML	CPU, link bandwidth	Coordinated	A RNN-based admission control to predict the feasibility of a VNE solution given historical experience records for reducing execution time in online VNE.	Non-adaptive model, stationary environments.
Blenk et al. [115]	ML	CPU, link bandwidth	Uncoordinated	A Hopfield network, a special artificial neural network, preselecting a subset of good substrate nodes for VNoM in VNE by extracting subgraphs.	Non-adaptive model, low efficiency and stationary environments.
Haeri and Trajkovic [116]	RL	CPU, link bandwidth	Uncoordinated	MDP model using the Monte Carlo tree search algorithm for VNoM in VNE.	Sub-optimal solutions, time complexity and stationary environments.
Authors in [19, 117]	RL	CPU, link bandwidth	Uncoordinated	RL model using a dynamic attribute matrix representation as the input of policy network in CNN and NN trained for minimizing cross-entropy loss function.	Stationary environments and sub-optimal solutions.
He et al. in [29]	RL	CPU, link bandwidth, energy	Uncoordinated	MDP model using an improved Q-Learning algorithm with a curiosity-driven mechanism	Local optima.
Wang et al. [30]	RL	CPU, link bandwidth	Uncoordinated	VNE modelled as MDP based on Temporal-Difference Learning with NN as an approximator.	Time complexity, stationary environments and no details in training and testing.
Blenk et al. [20]	RL	CPU, link bandwidth	Uncoordinated	A Continuous-Decision VNE scheme based on RL with a policy-gradient mechanism for updating RNN parameters.	Stationary environments and low efficiency.
Andreoletti et al. in [21]	RL	CPU, link bandwidth	Coordinated	A privacy-preserving (LID) Q-Learning algorithm for solving VNE problem across multi-domain infrastructure under the Shamir Secret Sharing scheme	A trade-off between overheads and embedding cost, stationary environments.
Dolati et al. [31]	DRL	CPU, link bandwidth	Coordinated	An adaptive Deep RL-based VNE approach using a convolutional deep neural network for automatically extracting features.	Nonscalability, stationary environments and inefficient reward function.
Rkhami et al. [33]	DRL	CPU, link bandwidth	Uncoordinated	An episodic MDP model based on DRL and GCN for training the agent.	Nonscalability, stationary environments.

Yan et al. [23], [24]	DRL	CPU, link bandwidth	Coordinated	Automatic VNE based on DRL with a parallel policy gradient training as well as multi-objective reward function, and GCN for automatically extracting spatial features of SN.	Expensive training, stationary environments.

Table 2.1: Taxonomy of VNE approaches

2.4 Distributed and Parallel Computing Framework

2.4.1 Introduction

Distributed and parallel computation is currently an intense research topic motivated by a need for a solution of immense computational problems. Recent technological advances and lower cost of powerful computing devices make the possibility of large-scale parallel computation and the solution of such problems possible. Moreover, the availability of powerful parallel computing devices and advanced technologies have generated more interest in new problems that were not dealt with before. There are several applications of parallel and distributed computation such as analysis, simulation, optimization of large-scale inter-connected systems; the solution of mathematical programming, optimization problems; and information acquisition/extraction and control in distributed systems. The major concerns of the above classes of applications are cost and speed. The hardware should not be very expensive while the computation should be done within an acceptable amount of time defined by particular applications. Besides cost and speed, reliable and precise operation of the proposed system in the presences of limited communication capabilities should be an additional concern.

Moreover, Artificial Intelligence (AI) applications play an important role in the development of this subject. Genetic Algorithm (GA), a subset of Evolutionary Computation algorithms, is generally considered as AI mechanism that has been proven its impressive efficiency on different fields. Motivated by these aforementioned facts, we come up with a distributed and parallel GA-based scheme for embedding virtualized NFs on the substrate network in VNE problem. By this proposal, our goals are not only to improve the embedding performance, but also significantly reduce the execution time compared to sequential paradigm.

2.4.2 Principles of Distributed and Parallel Computing

The starting era of computing was with the radical development of hardware architectures, specifically in advent of compilers and operating systems, supporting system management and application development [118].

Computing is converted into a service-provisioning model that users access available services based on their demands without considering where these services are hosted. Cloud computing is the most emerging infrastructure that brings computing utility into reality. It is a technological innovation that concentrates on the manner of designing computing systems, developing applications and leveraging providing services for constructing software. This is relied on the concept of dynamic provisioning that can be generally applied to services and physical infrastructure including computing capacity, storage and networking. The network resources are offered to users on a pay-per-use basic from the cloud vendors. It said, cloud computing provides renting paradigm, virtual hardware, run-time platforms and pay-per-use services and extends a flexible ability of integrating extra capacity or new features/services into the existing infrastructures.

In essence, clouds are considered as a large number of distributed computing facilities that offer available on-demand services to users. A distributed system comprised several independent components is recognized as a single entity by its users, that is also the characteristic of cloud computing. A distributed system characterization can be expressed as *A distributed system is a collection of independent computers that appears to its users as a single coherent system* [119]. The main goal of distributed systems is to enhance shared resource utilization. Distributed systems feature heterogeneity, scalability, concurrency, transparency, openness, availability and independent failures [118]. However, the terms parallel computing and distributed computing are usually utilized interchangeably, so an distinction between them is important.

The term of parallel computing term refers a tightly-coupled system while distributed computing term indicates a wider system class including those that are tightly coupled as well. Initially, parallel computing implies a computation model where the computation is partitioned among several processors that share the mutual memory. As such, a parallel program can be then broken down into multiple execution units allocated into several processors that can communicate each other through mutual memory in a single computer. Over time, this model is now extended to all

architectures that support parallelism in either hardware or software. An example of a cluster that includes a set of connected nodes communicating through a network and is setup with a distributed-shared memory system can be recognized as a parallel one.

In contrast, distributed computing involves architectures or systems that enable the computation to be broken down into smaller units and conducted on various computational elements concurrently. These elements are whether processors on different nodes or processors in the same computer, or cores of a processor. It means that the term distributed would signify that locations of computing elements are different and these can be heterogeneous towards hardware and software. A typical example of distributed computing systems is the Internet which altogether combines a large number of architectures, systems and applications. Thus, distributed computing that is often acknowledged as a general term is consisted of a wide range of systems and applications than parallel computing. Perhaps, distributing computing could improve the performance of a standalone application as if such application runs on a single machine, that needs to perform tedious calculations. By distributing these calculations to several machines, performance might be enhanced.

Hence, the term "distributed and parallel computing" can be preferable to the process that uses multiple computing devices to process the same complex task in parallel.

Traditional computing systems tend to deploy a centralized server for data processing, storing and retrieving. A large number of generated data cannot be accommodated by standard servers, causing extremely bottleneck while processing multiple tasks simultaneously. MapReduce technique [35] can solve such bottleneck issues when splitting the input dataset into independent chunks that are then handled by two crucial tasks: Map and Reduce in a entirely parallel manner. The former task performs filtering and sorting and its outcome becomes input of reduce task for summary operation. Both the input and the output of the tasks are typically stored in filesystem. The main advantage of MapReduce is ease of scaling data processing across multiple computing nodes. This simple scalability is also the most attractive feature of MapReduce model.

2.4.3 Metaheuristic Genetic Algorithm

An optimization process progressively discovers better solutions by searching and comparing feasible outcomes until it cannot reach out a better fitted solution [120]. The goal of optimization is to approach an optimal solution that meets multiple desired

objectives and a set of predefined rigid constraints. Evolutionary Computation (EC) in computer science is in the context of artificial intelligence (AI) strongly inspired by nature. EC is perceived as "the next major transition of artificial intelligence" or "The Next Big Thing" in AI fields [121] due to its several unprecedented benefits.

EC includes a set of metaheuristic algorithms that explore the global optimal solution for a specific problem. EC techniques are stochastic algorithms in which searching methods can model natural phenomena. Imitating the natural evolution processes, the idea of EC algorithms is mimicking what the nature is doing. A such, the computational models of EC algorithms can be considered as an extensive over-simplification of the biological process. Metaheuristics in EC that comprised several optimization techniques can deploy various operators to generate potential solutions, and then a selection mechanism drives these solutions to the optimum. metaheuristics can be characterized roughly as either single solution or population based. Metaheuristic algorithms have been successfully applied to a wide range of optimization problems (e.g. classification, regression, clustering, design, optimization, planning, computer-based program generation) that are arisen in different fields. As such, metaheuristics offer exceptional performance, flexibility and adaptability to solving a complex problem. Many of those problems are combinatorial optimization problems which are computationally hard, in a short term $\mathcal{NP}$-hard. This means that the computing time grows exponentially towards the size of the problem (e.g. travelling salesperson problem). The most common population-based metaheuristic algorithms can be listed here including Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), Harmony Search (HS) and Firefly Algorithm (FA). Except those, we also have artificial bee colony algorithm, cuckoo search, hunting search, memetic algorithm and many more. Interested readers might like to reference [122] for further information.

Genetic Algorithm is the most popular evolutionary algorithms regarding the diversity of its applications. It is a mature stochastic search algorithm that derives from Darwin's evolutionary principle of natural selection, proposed by John Holland in the 1960s. A conventional GA algorithm consisted of initialization, selection, crossover and mutation operations [123]. At every iteration, GA will randomly select from the current population parents who will produce children/offspring as the next generation. Over generations, the population evolves and the possibility of attaining an optimal solution is highly increased. The first step of GA is to generate an initial

population usually at random. The quality of each possible solution in the initial population, also known as a chromosome, is measured relied on its fitness function. A pair of chromosomes is then selected to be parents based on their fitness values. In a crossover operation, the selected parental chromosomes are able to produce offspring by exchanging partial genes at a random crossover point. More than one random crossover point can be employed depending on the chosen crossover strategy. The next GA operator is mutation, applying a small random tweak to a randomly selected chromosome to generate a new solution. Mutation is associated with exploration of the search space by maintaining diversity of the population. This is a critical factor that influences the convergence of the GA algorithm. GA operators should be repeated continuously until the predefined terminating condition is reached. Moreover, GA is a stochastic search algorithm that is often deployed in machine learning applications. This algorithm is important in this field due to three reasons: GA performs on discrete spaces in which gradient-based methods are inapplicable. The next reason is that GA is technically a form of reinforcement learning algorithms. Finally, GA associates with population or a group of entities (multi-agent systems) [124].

GA algorithm gains many technical advantages compared to the typical optimization algorithms. The two most notables include the ability to solve complex optimization problems and parallelism. GA is able to handle diverse types of optimization like whether stationary or non-stationary (changes over time), linear or non-linear, discrete or continuous, or with random noise. Moreover, due to the fact that multiple offsprings in population can be considered as independent agents, the corresponding population can simultaneously explore the searching space in many different directions. Such ideal feature can adopt the paralleled implementation of the algorithms. Besides great benefits, GA itself remains some inevitable drawbacks. We need to carefully figure out population size, selection strategy, the formulation of fitness function and the rates of crossover and mutation operators. Any improper selections will affect the convergence of GA algorithm or expected performance results. Despite these disadvantages, GA is still one of the most popular optimization metaheuristic algorithm in modern optimization [125].

GA is suitable for solving multi-objective linear or non-linear programming optimization problems due to its simplicity and ease of implementation. It is fast and more efficient than typical heuristic methods by balancing between exploration and exploitation by setting parameters properly. GA is advised as a scalable alternative to

the cutting-edge reinforcement learning (RL) algorithm with competitive performance results. Even GA does not always outperform RL algorithms in [126] and [127], but GA is considerably faster than RL when it exposed great scalability and parallel capabilities. In practise, GA can be fundamentally recognized as a parallel search [34] with no mutual dependency among several exclusively feasible solutions. Instead of achieving a single solution, GA provides a set of "good" solutions that can evolve over iterations driven by a fitness function.

However, GA algorithm based on randomness is not analytically perfect as it cannot always guarantee the best performance efficiency like heuristic algorithms. The base of GA's efficiency is relied on the assumption that the quality of selected parents will determine the quality of their children as natural selection theory. This hypothesis still holds up for now. In practise, GA produces excellent solutions in average and it is a practical choice in reality due to its simplification, ease of parallelism and good performance achieved.

GA algorithm is better than other metaheuristic algorithms due to the facts that similar to RL algorithms in the aspect that GA interacts with environments through an iterative action-reward process driven by a multi-objective fitness function. Crossover and mutation operations play the exploitation and exploration roles respectively that allow GA algorithm to explore the searching space efficiently.

2.5 Chapter Summary

In this chapter, we presented an introduction to virtual resource allocation, its problem descriptions and formulations. In online dynamic environment, time is critical so faster embedding algorithms can be practically applicable to real networks whereas the performance efficiency is realistically expected by any network operator. VNE algorithms that can pose a fast embedding capability while maintaining good efficiency are highly desirable. Therefore, the real world needs these practical algorithms meeting those goals.

From these starting points, we extensively studied the state of the art VNE research topic, and then categorized them into various categories. Furthermore, a new classification towards reinforcement learning approaches for solving virtual network assignments is also introduced. In essence, VNE approaches can be commonly classified into mathematical optimization methods, heuristics, metaheuristics and

finally machine learning-based algorithms.

Although mathematical algorithms can produce optimal solution in theory, they have a serious problem with time complexity, which prevents them from being applied to online VNE problem. In fact, there is no actual optimal solution for online VNRs since it is very difficult to take the uncertain future into account. Hence, most of typical VNE approaches concentrate on heuristic algorithms which could guarantee the VNs mapping speed, but they most likely suffer from sub-optimal solutions. Even heuristics have to scarify performance efficiency, they are still a preference in online embedding systems due to a fast mapping speed.

To achieve efficiency aspect while keeping the embedding speed within polynomial time, metaheuristic approaches, a subset of AI algorithms are widely proposed to improve the VNRs mapping efficiency since they allow the potential solutions to get evolved through intelligent evolvement operations. Although metaheuristic-based algorithms can somehow guarantee the efficiency, they are slower than heuristic algorithms in terms of the mapping speed. Fortunately, metaheuristic can be easily paralleled in nature so that an appropriate distributed and parallel approach can solve such slow speed problem.

Moreover, the majority of the aforementioned VNE approaches are only focusing on VNoM stage following the VLiM stage that is mainly relied upon either the shortest path methods (e.g., Dijkstra's) or MCF mechanism, where the shortest path is widely known as the most popular and fastest algorithm for VLiM stage.

Some ML-based approaches were proposed to solve the VNE problems. However, ML is not feasible for dynamic environments like online VNE applications where statistical behaviours of environments keep always changing. ML requires the labelled inputs and outputs for training models. Unfortunately, we have not had the outputs yet towards online VNRs. Thus, exact methods can be carried out to achieve the outcomes of the corresponding inputs, and then they can utilized for model training. Although, exact methods can approach optimal solutions for offline VNE problems, there are no actual solutions for online VNE when we do not have any information about the future requests and integrating the future into the mathematical optimization models is not an easy task. Therefore, ML, specifically supervised learning, can be applied for online VNE problem, but ML suffers from curse of dimensionality in training and the low accuracy when applying into online VNE.

Recently, RL has been merged with deep learning techniques as a promising

solution for solving a wide range of networking problems, but it is too difficult to apply RL techniques directly to online VNE problem due to its dynamic environment.

RL algorithms can provide a promising approach to solving a wide range of optimization problems. As mentioned in previous chapter, they face several fundamental challenges when being applied to online VNE applications. The challenges derive from the fact that RL algorithms are based on parametric models, which are highly dependent on stationary environments. A specific MDP model has to be trained for a specific environment. Thus, a large number of environment scenarios are associated with a large number of models. In fact, RL algorithms can work very effectively when environments are limited to few stationary scenarios. However, when dealing with very dynamic environments, they suffer from the curse of dimensionality that are very hard to solve.

Online VNE applications serve as an excellent example of such highly dynamic environments, where virtual network demands change dramatically at different time scales with different request rates, different requested topologies, and different amounts of requested resources for virtual nodes/links. If trained offline, numerous models have to be trained and stored beforehand for all environments that may happen. Furthermore, when applied to an online VNE application later, one of the pretrained models has to be selected based on the identification of the specific environment at the time, which is a time consuming process that makes it hard to meet the stringent delay requirements for online VNE applications. If trained online to remove the requirements for training numerous models beforehand, the long training time of RL algorithms still makes them hard to meet the stringent delay requirements for online VNE applications, which can create an awkward situation that the trained models may become out of date before they can be applied.

Our genetic algorithm based approach bears a resemblance to RL algorithms in the aspect that they both interact with environments through an iterative action-reward process. However, different from RL algorithms, our approach is not dependent on any specific models. It can be applied to nonstationary environments as well as stationary ones. It does not need any pretraining. It can meet the stringent delay requirements for online VNE applications by adopting our proposed distributed and parallel implementations. It can achieve better performances than many existing algorithms. Therefore, we again believe our approach is fast, efficient, and practical.

Consequently, we propose a distributed and parallel GA-based algorithm that is

not only faster than the well-known fastest heuristic approach, but also enhances VNE solutions in efficiency. Through this approach, challenges of online VNE problem raised in previous sections can be efficiently solved. We also express GA's terminologies, its operations and ideas of a distributed and parallel framework. Some weaknesses of typical GA algorithm are then specified. We apply our proposed approach in different domains including VLiM, joint node-link problem and online offloading task at the edge. Details of our proposed solution and its variances are described in next chapters.

Chapter 3

Intelligent Distributed And Parallel Resource-Allocation Algorithms

3.1 Introduction

Over the past decade, NV has received intense attention from both industry and academia since it is not only a key technology to deal with the ossification problem of current Internet architecture [1], but also an emerging paradigm for achieving future network architectures such as virtualized 5G network [7] and smart IoT networks [8]. NV maintains the diverse services and applications that provide various users through the sliceable management of existing physical network resources, and permits seamless sharing of physical network resources among multiple VNs. It also enables the isolated coexistence of multiple VNs on a substrate network (SN). This prevents unnecessary expansion of the existing infrastructure, reduces the wastage of substrate network resources and improves network utilization. VNE, the main obstacle to NV, enables multiple miscellaneous virtual networks to co-exist on top of the shared physical infrastructure. VNRs arrive dynamically and stay in the network for an arbitrary duration in most scenarios. Due to its complexity, a typical VNE process can be divided into two sub-problems: VNoM and VLiM. VLiM is more challenging than VNoM because of its rigorous requirements. Consequently, all substrate links of the potential link mapping solutions of a VN link request that a path traverses must have enough remaining capacity. This inevitably leads to bandwidth fragmentation. Moreover, mapping requested virtual links on the underlying shared physical infrastructure with several rigorous constraints is still $\mathcal{NP}$-Hard, even for offline embedding [43].

In dynamic environment of online VNE where VNRs dramatically change at

different time scales with different arrival rates, different topology, and different resources for virtual nodes and links, time is critical so that InPs attempt to map many VNRs as fast as possible while a large number of online VNRs keep arriving in the network. Consequently, practical VNE algorithms for mapping online VNRs must be enough fast while maintaining good performance efficiency.

Unfortunately, optimization models like Integer Linear Programming (ILP) are not appropriate for online VNE problems to obtain optimal solutions within polynomial time because of their complexity, nonscalability and time consumption. Thus, most of VNE solutions centralize heuristic algorithms since they are able to guarantee a fast embedding speed applicable to online VNE applications. However, heuristic approaches mostly likely fall into sub-optimal solutions, leading to low performance efficiency. On the other hand, metaheuristic approaches, a subset of AI algorithms can be considered to enhance the mapping efficiency since the potential solutions can be evolved after intelligent evolvement operations, that helps explore the search space efficiently. Even producing good efficiency, metaheuristics perform slower than heuristic algorithms regarding the embedding speed. To solve this problem, metaheuristic algorithms can be adopted in a distributed parallel implementation to improve the embedding speed. This adoption comes from the fact that the lower cost of computing hardware is beneficial to parallel algorithms in solving complex computation tasks and metaheuristics can be easily paralleled in nature.

Moreover, most research work [12, 19, 20, 29, 30, 43, 55, 58–60, 62, 68–71, 78, 84, 85, 90, 91, 93, 94, 101, 115–117] focuses only on achieving efficient node mapping, and leaves the link mapping stage for the k-shortest path method or multicommodity flow (MCF) mechanism. This definitely limits the VN link mapping selections and it seems that VLiM stage is being underestimated.

Together with fifth generation virtualized networking and IoT, VNE problems, where the substrate network supports either path splittable or unsplittable configuration, are a critical research topic in several research fields such as Future Edge Clouds, Software Defined Network (SDN) and Network Function Virtualization (NFV). Although splittable-enabled mapping achieves a better resource utilization in theory, the unsplittable mechanism can seem harder to approach optimal solutions than its splittable counterpart. A splittable-based solution creates an abundant overhead for consistently maintaining the network state [47] and possibly causes the problem of out-of-order package delivery with extra latency that might not be acceptable for

sensitive-delay applications. Therefore, we only consider unsplittable mappings in this dissertation.

In this chapter, we propose several GA-based algorithms for VLiM stage, being adopted in distributed parallel implementations to significantly reduce execution time while maintaining good performance efficiency. These are major contributions of this chapter:

1. We introduce a generic distributed and parallel GA-based framework for virtual resource allocation problems. We also provide an analytical study on execution time under the parallel scheme. Our evaluation has illustrated that the proposed GA-based algorithm can be solvable in logarithmic time.

2. We then propose an intelligent algorithm [13] that exploits distributed parallel machines, dealing with online VNE link mapping problems. In GA, a chromosome is defined as a link mapping solution of all virtual link requests of a given VNR, and each gene within a chromosome is the mapping solution of a virtual link. The proposed GA-based algorithm is guided by an nonlinear fitness function that simply takes hop-count and bandwidth into account, which makes our approaches different from others. Our approach considerably reduces the execution time while achieving high performance efficiency.

3. Expanding upon [13], we increase the complexity of the fitness function by introducing three more evaluation factors which are expected to improve performance results. To the best of our knowledge, this is the first research that presents attraction strength as a factor in fitness function for VNE problems [128].

4. Moreover, we propose a new crossover operation that can be applicable to VLiM, namely, elastic crossover [129]. In new crossover, the random number of genes are exchanged between the selected parents to generate new generations.

5. Previous work has used a simple greedy method for VNoM stage. To enhance efficiency, we propose a combination between the node ranking mechanism, that considers several network topology attributes and global network resources, for VNoM stage and our distributed parallel GA-based algorithm for VLiM.

6. We have proved that VLiM plays an essential role in dealing with online VNE problem, which is usually abandoned in most of research papers where VLiM is commonly assigned to the shortest path method due to its simplicity and rapid

speed. Our proposed algorithms for VLiM are not only faster than the fastest and most popular heuristic VNE algorithm for more than 40% by adopting distributed parallel implementations, but also perform better than many VNE algorithms in several performance metrics.

3.2 Network Model and Problem Descriptions

3.2.1 Virtual Network Assignment

A substrate network is modelled as a weighted undirected graph and denoted as $G^s = (N^s, L^s)$, where N^s is the set of substrate nodes and L^s is the set of substrate links. Undirected graph is chosen for formulating the VNE problem since they can simply facilitate the model descriptions and directed graphs do not have any impacts on our approaches. Each substrate node $n^s \in N^s$ has a geographic location $loc(n^s)$, also associated with availability of CPU capacity denoted by $c(n^s)$. Likewise, a substrate link $l^s \in L^s$ of two substrate nodes has the corresponding bandwidth capacity $b(l^s)$. Similarly, the i^{th} arriving VN request can be modeled as a weighted undirected graph that is denoted by $G_i^v = (N_i^v, L_i^v)$, where N_i^v and L_i^v are the set of virtual nodes and links of the i^{th} VN request, respectively. Each virtual node $n_i^v \in N_i^v$ is associated with a CPU requirement $c(n_i^v)$, while each virtual edge $l_i^v(s_i^v, d_i^v) \in L_i^v$ between the corresponding virtual source s_i^v and destination nodes d_i^v has a bandwidth requirement $b(l_i^v)$. Each VNR has a mapping radius $D(n_i^v)$ expressing how far a virtual node n_i^v can be placed from the location identified by $loc(n_i^v)$. Without loss of generality, memory and storage are not taken into account in this dissertation due to the fact that memory or storage is similar to CPU or bandwidth resources, so removing them in the model does not affect the embedding results but can simplify the VNE model.

Embedding a VNR G_i^v onto the corresponding substrate network G^s can be divided into two major components: VNoM and VLiM. In a node embedding stage, a virtual node from the same VN request can be embedded onto a single substrate node $\mathcal{A}_\mathcal{N} : N_i^v \rightarrow N^s$, with $n^v \in N_i^v$ subject to:

$$R_N(\mathcal{A}_\mathcal{N}(n_i^v)) \geq c(n_i^v) \tag{3.1}$$

$$D(n_i^v) \geq \mathcal{D}(loc(n_i^v), loc(\mathcal{A}_\mathcal{N}(n_i^v))) \tag{3.2}$$

$$\mathcal{A}_{\mathcal{N}}(n_i^v) \in N^s \tag{3.3}$$

$$R_N(n^s) = c(n^s) - \sum_{n^v \to n^s} c(n_i^v) \quad , \tag{3.4}$$

where $n^v \to n^s$ indicates the virtual node n^v hosted on the substrate node n^s. $\mathcal{D}(i^s, j^d)$ expresses the distance between two nodes i^s and j^d, whereas $R_N(n^s)$ is the residual CPU capacity of a substrate node. Each virtual link can be embedded onto a substrate path (unsplittable) with one or more substrate links. This is defined by $\mathcal{A}_{\mathcal{L}} : L_i^v \to L^s$, with $l_i^v = (s_i^v, d_i^v) \in L_i^v$. $\mathcal{E}^s(\mathcal{A}_{\mathcal{L}}(l_i^v))$ is a possible set of all physical paths from source node $\mathcal{A}_{\mathcal{N}}(s_i^v)$ to destination node $\mathcal{A}_{\mathcal{N}}(d_i^v)$:

$$\mathcal{A}_{\mathcal{L}}(s_i^v, d_i^v) \subseteq \mathcal{E}^s(\mathcal{A}_{\mathcal{N}}(s_i^v), \mathcal{A}_{\mathcal{N}}(d_i^v)) \tag{3.5}$$

subject to:

$$R_L(e^s) \geq b(l_i^v), \forall e^s \in \mathcal{E}^s(\mathcal{A}_{\mathcal{L}}(l_i^v)) \tag{3.6}$$

$$R_L(e^s) = \min_{l^s \in e^s} R_L(l^s) \tag{3.7}$$

$$R_L(l^s) = b(l^s) - \sum_{l_i^v \to l^s} b(l_i^v) \quad , \tag{3.8}$$

where $R_L(e^s)$ is the available bandwidth of the physical path $e^s \in \mathcal{E}^s$, while $R_L(l^s)$ is the residual capacity of a physical link.

3.2.2 Performance Metrics

In this research thesis, revenue is recognized as the sum of the total virtual resources embedded onto the corresponding substrate network over time.

Revenue of the i^{th} VNR G_i^v is calculated below:

$$R(G_i^v) = w_b * \sum_{l_i^v \in L_i^v} b(l_i^v) + w_n * \sum_{n_i^v \in N_i^v} c(n_i^v) \quad , \tag{3.9}$$

where $b(l_i^v)$ is the bandwidth requirement of the virtual link l_i^v, and $c(n_i^v)$ is the CPU requirement of the virtual node n_i^v. w_b and w_n are the unit weights of the mapped bandwidth and CPU resources, respectively.

Cost is the total substrate network resources that have been already allocated to the i^{th} virtual network.

$$C(G_i^v) = \sum_{n_i^v \in N_i^v} c(n_i^v) + \sum_{l_i^v \in L_i^v} \sum_{l^s \in L^s} f_{l^s}^{l_i^v} \quad , \tag{3.10}$$

where $f_{l^s}^{l_i^v}$ denotes the bandwidth of the substrate link l^s allocated to the virtual link l_i^v.

Revenue to Cost ratio (R/C) is defined as the ratio of average embedding revenue over average embedding cost, which can be used to evaluate the efficiency of a VNE algorithm. The revenue factor alone does not determine the efficiency of a VNE solution since it reflects only the successfully allocated VNRs while lacking cost information of corresponding VN embedding solutions. Maximizing revenue associated with the acceptance ratio while minimizing VNE costs maximizes the generated profit of InPs. A higher acceptance ratio with a lower average revenue to cost ratio is an inadmissible result, and indicates that the SN resources have been inefficiently utilized. As the result, the revenue to cost ratio should be considered simultaneously with the acceptance ratio [116].

$$\mathcal{RC}(G_i^v) = \frac{R(G_i^v)}{C(G_i^v)} = \frac{w_b * \sum\limits_{l_i^v \in L_i^v} b(l_i^v) + w_n * \sum\limits_{n_i^v \in N_i^v} c(n_i^v)}{\sum\limits_{n_i^v \in N_i^v} c(n_i^v) + \sum\limits_{l_i^v \in L_i^v} \sum\limits_{l^s \in L^s} f_{l^s}^{l_i^v}} \quad . \tag{3.11}$$

In fact, revenue and cost ratios can be used separately to evaluate an efficiency of an VNE algorithm. In that case, we need to take a look at different indexes and then manually combine those information to point out the conclusion. More algorithms we compare, more results we need to concern. Instead of these cognitive/computational steps, R/C ratio is able to directly provide an accurate assessment on these outcomes.

Acceptance ratio is defined as the ratio between the number of accepted VNRs over arrived VNRs during the interval time τ as computed below:

$$A_c^\tau = \left| \frac{\xi^a(\tau)}{\xi(\tau)} \right| \quad , \tag{3.12}$$

where $\xi^a(\tau)$ and $\xi(\tau)$ denote the successfully embedded VNRs and the number of arrived VNRs respectively.

Node utilization reflects the distribution of network loads on the corresponding SN. Node utilization is derived from the amount of network resources occupied by virtual node requests during a certain time, divided by the total amount of node

resources. Node utilization can be expressed as follows:

$$\mathcal{U}_N(N^s) = \sum_{n^s \in N^s} \left(\frac{\sum_{n_i^v \to n^s} c(n_i^v)}{c(n^s)}\right) * T_i \quad , \tag{3.13}$$

where T_i represents the duration of the accepted i^{th} VNR.

Link utilization Similarly, link utilization is derived from the amount of network resources occupied by virtual link requests during a certain time, divided by the total amount of link resources. Link utilization is defined as follows:

$$\mathcal{U}_L(L^s) = \sum_{l^s \in L^s} \left(\frac{\sum_{l_i^v \to l^s} b(l_i^v)}{b(l^s)}\right) * T_i \quad , \tag{3.14}$$

Remaining bandwidth is the remaining bandwidth of a substrate network that can be calculated as follows:

$$\mathcal{R}_m(L^s) = \sum_{l^s \in L^s} \left(b(l^s) - \sum_{l_i^v \to l^s} b(l_i^v)\right) \quad . \tag{3.15}$$

When new VNRs arrive, an InP first needs to calculate remaining network resources and then attempt to map the VNRs onto the substrate network. After the virtual nodes have been mapped, embedding virtual link requests depends mainly on the remaining bandwidth capacity of the physical network. Higher remaining bandwidth increases the possibility of new VNRs being accepted.

VNE problem in this dissertation involves several performance metrics such as acceptance ratio, embedding cost, revenue, resource utilization that might be independent from the computer language used. However, in terms of execution time, it highly depends on the programming language or computer capacity by nature.

3.2.3 Objectives

The main interest in this dissertation, specifically in Chapter 3 and 4 is to propose an elastic and agile operation scheme relied on an efficient online VN embedding algorithm. Similar to several papers, for example [12], we would like to increase the accumulated revenues while reducing the embedding cost towards InPs in long-run, in addition to improving link utilization of the substrate networks.

In fact, revenue factor can provide an insight into how much an InP gains by accepting a VNR, but this metric would not very helpful without quantifying the cost that InP needs to pay for embedding such request. The ratio between revenue and cost (R/C) will reflect how efficient the network resources are being utilized. Associated with acceptance ratio, high revenue is mostly desirable but if R/C ratio is low which means that the cost incurring for embedding VN requests is unexpectedly high due to inefficient embedding algorithms. In this case, the generated profits of corresponding InP are not maximized. If we only dive in the performance metrics related to VNoM (e.g., node utilization) to examine the network, it could not be enough. Consequently, network resources related to substrate links should be considered such as link utilization, remaining bandwidth. A such, more residual bandwidth increases the possibility of accepting future VNRs and less substrate resources are consumed for mapping them, equivalent to efficient VN mapping algorithm. The formulations of the aforementioned performance metrics are defined in Section 3.2.2

3.3 Distributed and Parallel GA-based Operation Scheme

3.3.1 Our Proposed Computing Paradigm

Imitating natural selection, GA is a compelling AI approach to address the complexity of both constrained and unconstrained optimization problems. In practice, GA can be considered as a parallel search [34] with no mutual dependency among several exclusively feasible solutions. Furthermore, parallel computing has recently emerged as a promising paradigm to solve complicated problems by enabling concurrency with time saving and low cost guarantees. Inspired from these facts, we come up with a distributed parallel operation scheme where GA algorithm is the core for solving online VNE problem with respect to both fast embedding speed and performance efficiency.

The generic design of our proposed GA-based algorithm being operated on a predefined number of independently distributed machines is depicted in Fig. 3.1. As illustrated, we specify the procedures that are consecutively working under a single master node such as node mapping and synchronization. In contrast, several working nodes independently run the proposed algorithm to discover as many feasible solutions

as possible for resource allocation problem. Each independent working machine intentionally operates with a number of predefined iterations to find feasible virtual link embedding solutions. A synchronization step then selects the ideal one among several solutions detected by the distributed working nodes. Unlike other previous approaches, our proposed algorithm aims to embed multiple resource allocation requests in corresponding VN request at once instead of utilizing sequential embedding.

In fact, other heuristic/metaheuristic algorithms can be implemented in parallel, which significantly depends on their implementation complexity and the dependencies between their algorithmic operations. Due to mutually independent iterations and ease of implementations, GA is selected to solve online virtual resource allocation problems by adopting parallel implementations in this dissertation.

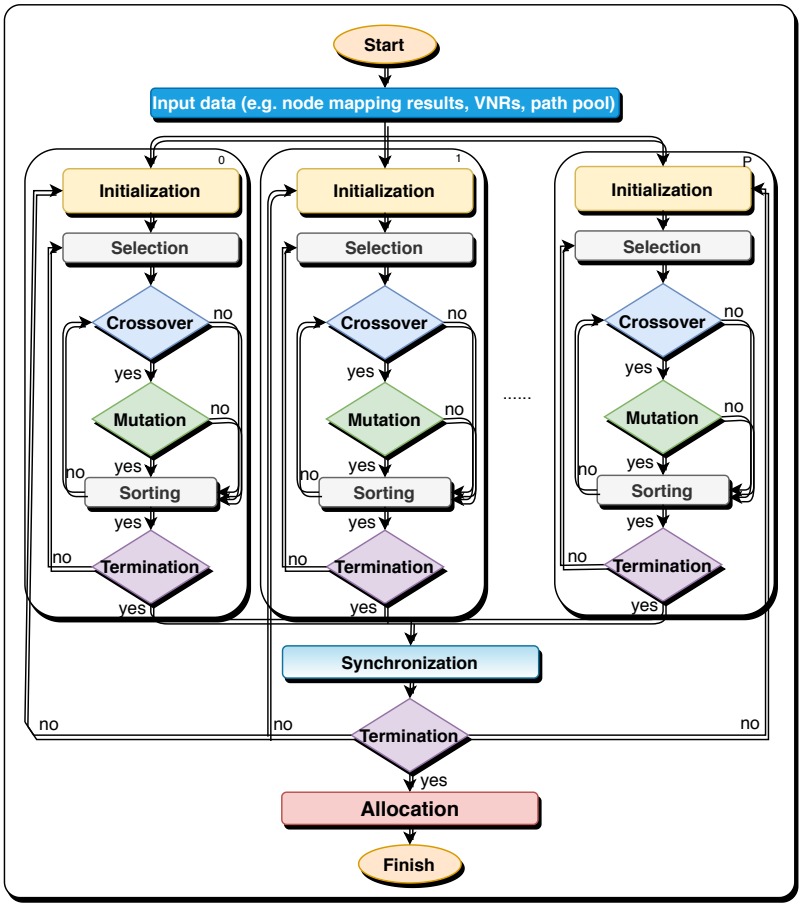


Figure 3.1: Parallel distributed operation framework

3.3.2 Execution Time Analysis

It is commonly observed that the uncertainty component increases the complexity of operation time calculation. As explained in previous sections, we propose a distributed and parallel computation model aimed at substantially reducing the CPU execution time. We denote $\mathcal{T}_m^i$, $\mathcal{T}_n^i$, $\mathcal{T}_o^i$, $\mathcal{T}_s^i$ and $\mathcal{T}_a^i$ as the operation time of the master node, node mapping procedure, path pool generator, synchronization phase and eventually VNR allocation, respectively. i^{th} denotes the current VN request's index, while $\mathcal{T}^i$ is the operation time of the paralleled working nodes. $\mathcal{T}^i$ is subject to the execution time of the last node to complete its assigned task. The parallel level denoted as p is a trade-off between availability of network resources and CPU completion time.

We define the operation time of a related sub-working node as $\mathcal{X}_t$ where t^{th} is the index $t \in [1, p]$ of the working node. Therefore, $\mathcal{T}_m^i$ and $\mathcal{T}^i$ are computed following:

$$\mathcal{T}^i = max\{\mathcal{X}_t\},\ t \in [1, p] \tag{3.16}$$

$$\mathcal{T}_m^i = \mathcal{T}_n^i + \mathcal{T}_o^i + \mathcal{T}^i + \mathcal{T}_s^i + \mathcal{T}_a^i \tag{3.17}$$

Similarly, $\mathcal{S}^i$ is denoted as the sequential execution time of the associated working nodes and can be computed as follows:

$$\mathcal{S}^i = \sum_{t=1}^{p} \mathcal{X}_t \tag{3.18}$$

$$\mathbb{E}[\mathcal{S}^i] = p\mathbb{E}[\mathcal{X}_t] \tag{3.19}$$

In a sequential scenario, $\mathcal{T}_m^i$ can be calculated as:

$$\mathcal{T}_m^i = \mathcal{T}_n^i + \mathcal{T}_o^i + \mathcal{S}^i + \mathcal{T}_s^i + \mathcal{T}_a^i \tag{3.20}$$

Each mutually independent working node is working in a parallel paradigm with an equal probability distribution. Consequently, its operation time is assumed to be independent identically-distributed random variable (iid). Considering the statistical performance results in Section 3.4.9, $\mathcal{X}_t$ with Normal Inverse Gaussian (NIG) as confirmed in Fig. 3.8a for example, we make use of the Chernoff-Cramer method,

taking advantage of the Moment Generating Function (MGF) to extract the upper tail bound of $\mathcal{T}^i$. As a consequence, the MGF of $\mathcal{X}_t$ is calculated as:

$$M_{\mathcal{X}}(z) = \mathbb{E}[e^{z*\mathcal{X}_t}] = e^{\left[z*\mu+\delta(\sqrt{\alpha^2-\beta^2}-\sqrt{\alpha^2-(\beta+z)^2})\right]} \tag{3.21}$$

where $\mu, \delta, \alpha, \beta > 0$
Using a portion of Jensen's inequality equation [130], we have:

$$\begin{aligned} e^{z*\mathbb{E}[\mathcal{T}^i]} &\leq \mathbb{E}[e^{z*\mathcal{T}^i}] = \mathbb{E}[max_t\{e^{z*\mathcal{X}_t}\}] \\ &\leq \sum_{t=1}^{p} \mathbb{E}[e^{z*\mathcal{X}_t}] = p\mathbb{E}[e^{z*\mathcal{X}_t}] \end{aligned} \tag{3.22}$$

When we take the logarithm of both sides, we have:

$$\begin{aligned} \mathbb{E}[\mathcal{T}^i] &\leq log(p)\left[\mu + \frac{\delta}{z}(\sqrt{\alpha^2-\beta^2} - \sqrt{\alpha^2-(\beta+z)^2})\right] \\ &= log(p)\left[\mu + \frac{\delta\sqrt{\alpha^2-\beta^2}}{z}(1 - \sqrt{1 - \frac{z^2+2\beta z}{\alpha^2-\beta^2}})\right] \end{aligned} \tag{3.23}$$

Assuming that $z, \mu, \alpha, \beta, \delta$ are constant values, we observe that the propensity of operation time under the parallel operating scheme is increasingly logarithmic. Compared with the linear propensity of the sequential model in (3.19), we can conclude that the proposed GA-based algorithm saves significantly more time when the parallel level p is increased. If K denotes the number of VN requests while $\mathbb{T}_g$ is the total execution time for allocating the arrived VN requests, then $\mathbb{T}_g$ can be computed as following:

$$\mathbb{T}_g = \sum_{i=1}^{K} \mathcal{T}_m^i \tag{3.24}$$

Total execution time of the entire scheme is approximated by mainly measuring the execution time of the worker node that produces the latest VN embedding solution. Then, this time measurement is added up with the operation time of the master procedures. Roughly, we can perceive that the proposed distributed and parallel embedding scheme is able to reduce the execution time from linear to logarithmic running time $O(log(p))$.

The generic framework and its operation time model will be applied for all our

VNE solutions in Chapter 4 and Chapter 5 in this thesis for approaching efficient VNE solutions and calculating the total execution time of our proposed algorithms.

3.4 An Intelligent Parallel Algorithm for Online VNE

As mentioned in previous sections, time is critical in online VNE problem so that we desire a fast and efficient VNE algorithm. Thus, we present an intelligent parallel algorithm based on GA that can be running on a predefined number of distributed machines in which they are independently operating to generate feasible solutions denoted as chromosomes. We define the procedures, which are functioning sequentially as working under a single master node such as node mapping, synchronization, etc,. while the ones independently operating parallel GA algorithms to find feasible solutions for VNE link mapping are working as several working nodes. Each working machine is independently running with defined iterations, the 'best' feasible VLiM solution will be selected among parallel machines. Our proposed parallel GA scheme is present in Fig 3.2 which is different from 3.1 towards two additional function blocks including node mapping and original path pool generation. Unlike other research papers that sequentially map requested virtual links, our proposed algorithm enables to map multiple link requests at once. A chromosome $\mathcal{C}_f$ is associated with a feasible link mapping solution of a VN. Whilst a gene g_i^j indicates a substrate path where i and j denote its current chromosome and virtual link respectively.

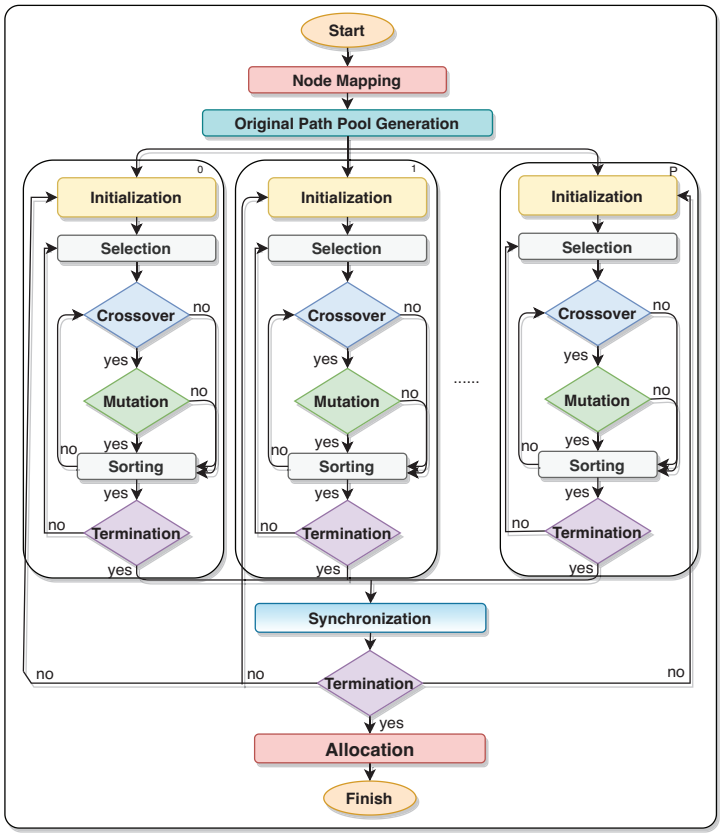


Figure 3.2: *Parallel operation scheme*

3.4.1 Node Mapping Algorithm

In contrast to research that utilizes exact formulation solutions that are computationally very expensive but may not guarantee optimal solutions, we deploy a Greedy mechanism for the node mapping stage similar to [43] in this research. Our motivation for selecting Greedy node mapping is to maximize remaining substrate network resources, reducing possible bottleneck issues of the substrate network. This strategy not only increases the possibility of accepting more incoming virtual node requests in the future, but also guarantees rapid VNE node mapping due to its simplicity. As cited in previous sections, we argue that both node and link mapping play equally key roles in producing an efficient VNE solution. The rapid speed of the G-SP algorithm in general and the shortest path algorithm in particular can be significantly exceeded with a novel approach we have urged.

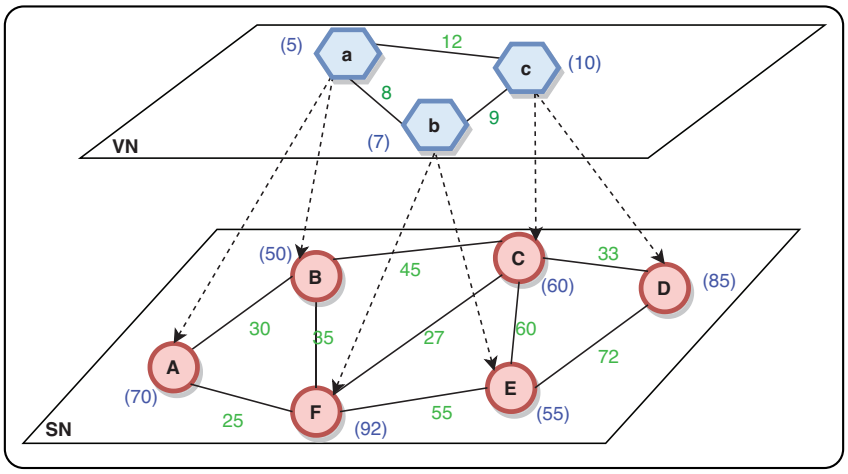


Figure 3.3: *An example of a VN mapped into a substrate network*

3.4.2 Genetic Representation

Normally, a VNR involves several virtual node and link requests, so the VNE problem commonly becomes multiple link embedding problems following node mapping. Instead of sequentially embedding each virtual link like most research approaches in this field, our proposed VNE algorithm permits embedding of several links at once. Each chromosome C_f, where f indicates the fth element in the initial population, represents a feasible embedding solution for all virtual link requests in a VNR in which each link mapping solution of a virtual link request is considered to be a gene. A gene g_f^j expresses a substrate path in accordance with a virtual link solution including a set of physical nodes and links. The subscript denotes its chromosome while the superscript indicates a jth virtual link in the chromosome. We also denote source, destination nodes and the corresponding substrate path between them as s_j^s, d_j^s and e^s, respectively. The gene can be encoded by (3.26) and (3.27). A substrate node can also be denoted as n_{jk}^s in which j and k indicate the gene and its position in the gene, respectively.

An example of gene encoding from the substrate node A to node D as depicted in Fig. 3.3 can be A-F-C-D where a physical path $e^s(A, D)$ includes three substrate links $l^s(A, F)$, $l^s(F, C)$, and $l^s(C, D)$.

Following set out is an actual example of the VNE problem. A VNR has three virtual nodes and three virtual links, as shown in Fig. 3.3. One possible solution is that virtual nodes a, b, and c can be assigned to physical nodes A, E and D,

respectively. Therefore, virtual link requests such as a to b, b to c and a to c become an embedding task to find feasible substrate paths from A to E, E to D and A to D, respectively in the link mapping stage. As a result, the chromosome comprises three genes: $e^s(A, E)$, $e^s(E, D)$ and $e^s(A, D)$. A simple example of chromosome encoding of the corresponding VNR can be expressed as {{A-B-F-E},{E-D},{A-F-C-D}}.

$$\mathcal{C}_\{ = \{g_j^1, g_j^2 \ldots, g_j^N\}, \tag{3.25}$$

$$g_j^f = \{e^s(s_j^s, d_j^s)\}, \tag{3.26}$$

$$g_j^f = \{n_{j1}^s, n_{j2}^s \ldots, n_{jk}^s, \ldots, n_{j\mathcal{D}_{s_j^s d_j^s}}^s\}, \forall k \in (1, \mathcal{D}_{s_j^s d_j^s}), \tag{3.27}$$

where

$$s_j^s = n_{j1}^s,\ d_j^s = n_{j\mathcal{D}_{s_j^s d_j^s}}^s. \tag{3.28}$$

3.4.3 Initial Path Pool Generation

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

3.4.4 Working Nodes

Population Initialization: each working machine starts to operate GA algorithm with a population initialization step. Each chromosome $\mathcal{C}_f$ represents a feasible solution. Assume that there are N genes and M chromosomes, an initial population

$\mathcal{P}$ (MxN size) at the machine k^{th} can be described as below:

$$\mathcal{P} = \begin{bmatrix} \mathcal{C}_1 \\ \mathcal{C}_2 \\ \vdots \\ \mathcal{C}_f \\ \vdots \\ \mathcal{C}_M \end{bmatrix} = \begin{bmatrix} g_1^1 & \cdots & g_1^j & \cdots & g_1^N \\ g_2^1 & \cdots & g_2^j & \cdots & g_2^N \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ g_f^1 & \cdots & g_f^j & \cdots & g_f^N \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ g_M^1 & \cdots & g_M^j & \cdots & g_M^N \end{bmatrix} \tag{3.29}$$

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

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

which relies on the cumulative sum of the fitness relative weights (3.33).

$$\mathcal{P} = \begin{bmatrix} \mathcal{C}_1 \\ \vdots \\ \mathcal{C}_s \\ \vdots \\ \mathcal{C}_r \\ \vdots \\ \mathcal{C}_M \\ \mathcal{C}_{M+1} \\ \mathcal{C}_{M+2} \end{bmatrix} = \begin{bmatrix} g_1^1 & \cdots & g_1^{j^c} & g_1^{j^c+1} & \cdots & g_1^N \\ \vdots & \ddots & \vdots & \ddots & \vdots & \\ g_s^1 & \cdots & g_s^{j^c} & g_s^{j^c+1} & \cdots & g_s^N \\ \vdots & \ddots & \vdots & \ddots & \vdots & \\ g_r^1 & \cdots & g_r^{j^c} & g_r^{j^c+1} & \cdots & g_r^N \\ \vdots & \ddots & \vdots & \ddots & \vdots & \\ g_M^1 & \cdots & g_M^{j^c} & g_M^{j^c+1} & \cdots & g_M^N \\ g_s^1 & \cdots & g_s^{j^c} & g_r^{j^c+1} & \cdots & g_r^N \\ g_r^1 & \cdots & g_r^{j^c} & g_s^{j^c+1} & \cdots & g_s^N \end{bmatrix} \tag{3.30}$$

Crossover: combines two parent chromosomes to form new offspring of the next generation. We denote $\mathcal{C}_s$ and $\mathcal{C}_r$ as two parent chromosomes with their indexes s and r in initial population. Moreover, assume j^c is the random crossover point between any genes inbound the N length, and new descendant chromosomes are denoted as $\mathcal{C}_{(M+1)}$ and $\mathcal{C}_{(M+2)}$ respectively. The offsprings are produced by swapping two parent genes starting from the crossover point j^c+1 to the end as depicted in 3.30. Obviously, the quality of generated children can be worse than whose ancestors or the duplication of solutions may happen at different parallel levels. Crossover point j^c is randomly chosen, but should not start at the first or last gene in the parents' chromosomes because the mated children are apparently same as their parents.

Mutation: applies random changes to individual parents to form new offspring. Mutation operation then includes randomly generating the mutation point denoted as j^m , and at that point, a new gene can replace the existing one of the in-processed chromosome to generate a new child. New selected gene which has been chosen from the original path pool must pass a feasibility check. Mutation is fundamentally implemented in the purpose of sampling the solution space and broadening the search. This is a crucial piece of the solution process that can somehow prevent from falling into the local optima. Assume j^m is denoted as mutation point while $g_{r'}^{j^m}$ is a new gene that replaced the existing one in $\mathcal{C}_{(M+1)}$. The mutation solution $\mathcal{C}'_{(M+1)}$ after

such substitution is $\mathcal{C}'_{(M+1)} = [g_s^1 \cdots g_{r'}^{j^m} \cdots g_s^N]$. Graphical illustration of crossover and mutation operators is shown in Fig. 3.4.

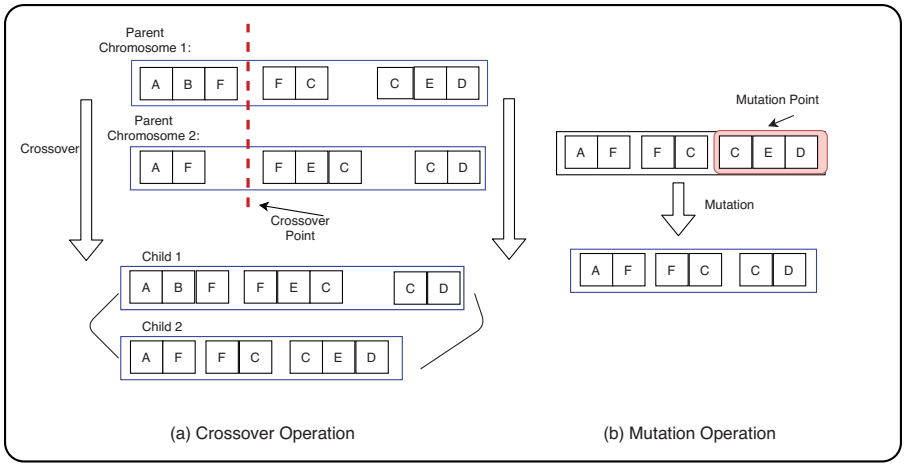


Figure 3.4: *An example of crossover and mutation operators*

3.4.5 Solution Sorting and Terminations

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

3.4.6 Synchronization and VNR Allocation

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

Algorithm 1 Proposed Intelligent VNE Algorithm

Input:
    i^{th} Virtual Network Request: $G_i^v = (N_i^v, L_i^v)$
Output:
procedure NODE MAPPING
    Sort virtual node mapping order
    for $n_i^v \in N_i^v$ **do**
        Select substrate nodes that meets location constraints - $\mathcal{D}(loc(n_i^v), loc(\mathcal{A}_{\mathcal{N}}(n_i^v))) \leq D(n_i^v)$
        Map n_i^v *to a substrate node* $\mathcal{A}_{\mathcal{N}}(n_i^v) \in N^s$ *with preferably higher resources* $R_N(n^s)$
        if (no available substrate node found) **then**
            return *Mapping failed*
    go to *link mapping phase*
procedure LINK MAPPING
    Generate the original path pool
    Implement **Genetic Algorithm** *in distributed parallel nodes*
    Synchronize achieved solutions among parallel nodes
    Select the best mapping solution based on fitness values
    if (no available substrate path found) **then**
        return *Mapping failed*
if *(Both mapping phases succeeded)* **then**
    return *(i^{th} VNR's solutions)*
    Update substrate network usage

3.4.7 Simulation Setup

We developed a discrete-event simulator in C++ programming language to evaluate the efficiency of our proposed algorithm with parameters similar to those in [12]. SN and VN topology are generated utilizing GT-ITM which is one of the most common topology generators in this field. In our simulation, the SN is set up with 50 nodes, randomly placed on a 25×25 Cartesian plane, connected randomly along 141 edges following the Waxman 2 model with $\alpha = 0.5$ and $\beta = 0.2$. The parameter α indicates maximal edge probability and β expresses edge length. Additionally, the CPU and bandwidth resources of the substrate network are uniformly generated in a range of [50-100]. Virtual network requests arrive in the network following a Poisson process at an average rate of between 4 and 8 virtual networks per 100 time units. In fact, time unit is relatively associated with the average workload. Poisson process is commonly

used to model the number of occurrences of random events (e.g., VNRs) in simulation due to its feature that the number of events occurred in a given interval are independent of the number in any other disjoint interval, which facilitates the event simulation. The lifetimes of VNRs follow an exponential distribution with an average of $\mu = 1000$ time units. The virtual nodes for each VN graph are configured by a random uniform distribution between 2 and 10, with average connectivity at 50%.

Basic CPU and bandwidth requirements generated for the virtual nodes are defined as the real random numbers that are uniformly distributed between [0-20] and [0-50], respectively. Each performance simulation ran for $50,000$ time units, 50 times longer than the average lifetime of a VN, to generate a large number of independent samples. All performance figures based upon average values are plotted with a 95% confidence interval.

3.4.8 Compared Methods

To evaluate the performance of our proposed GA-based algorithm, we compare it with several competitors as listed in Table 3.1. We selected DViNE, RViNE and G-SP [12] for our evaluation comparison for two reasons: performance and speed. While DViNE and RViNE are considered the highest performers because of their linear programming approach for node mapping, G-SP, which exploits the shortest path method for link mapping, is considered to be the fastest algorithm because of its simplicity. The shortest path method is broadly utilized by many heuristic and metaheuristic algorithms. To ensure a fair comparison, we adopted the node mapping algorithm used by G-SP. It is expected that our proposed algorithm will outperform DViNE and RViNE and be significantly faster than G-SP as a result of its enabled parallel processing design. Moreover, [12] is one of the most cited research papers in the VNE field [39]. This work provides an excellent embedding performance which serves as a benchmark for many other research papers [39]. Also, its source codes are publicly available, allowing researchers to reproduce accurate evaluation results.

Our simulation is conducted on a Ubuntu 19.10 (Eoan Ermine) 64-bit platform with 15.5 GiB memory and Intel Core i5-6200U CPU@2.30GHz×4.

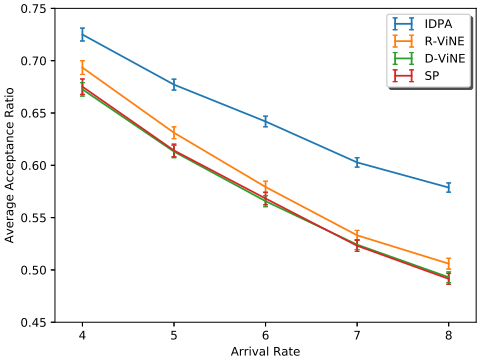
Table 3.1: *Compared Algorithms*

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

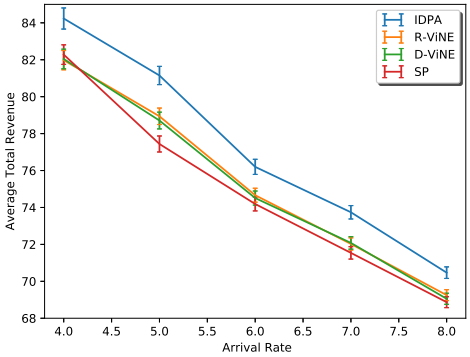
3.4.9 Execution Time Analysis

It is commonly observed that the uncertainty component increases the complexity of operation time calculation. As explained in previous sections, we propose a distributed and parallel computation model aimed at substantially reducing the CPU execution time. Its execution time analysis is detailed in Section 3.3.2.

To study execution time distribution, we essentially collect manifold operation time of the parallel procedures at several working nodes. Following the $\mathcal{X}_t$ histogram of IDPA as shown in Fig.3.8a, $\mathcal{X}_t$ basically fits Normal Inverse Gaussian distribution (NIG) that is defined as a normal variance-mean mixture where the mixing density is the inverse Gaussian distribution. The sum of square error of its fitting measured is $9.32e^{-3}$. The sample mean and standard derivation values of $\mathcal{X}_t$ of IDPA are 2.0363ms and 1.478ms respectively. Moreover, the average total execution time of different algorithms is demonstrated in Fig.3.8b in which the parallel level p of IDPA algorithm is set to 16. We determine the $p = 16$ since we recognized that the overall performance of IDPA after numerous experiments achieves convergence.

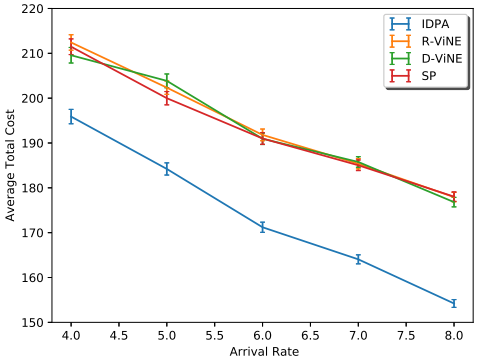


(a)

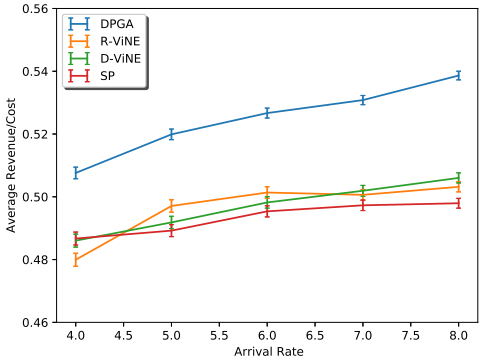


(b)

Figure 3.5: **(a)** VNR Acceptance Ratio **(b)** Average generated revenue

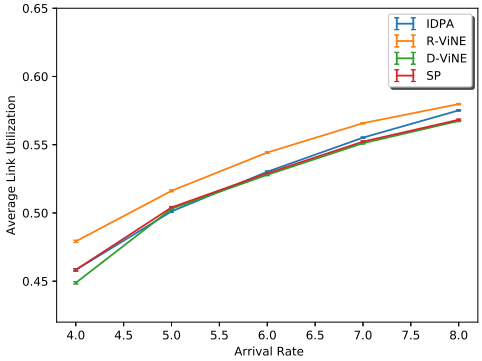


(a)

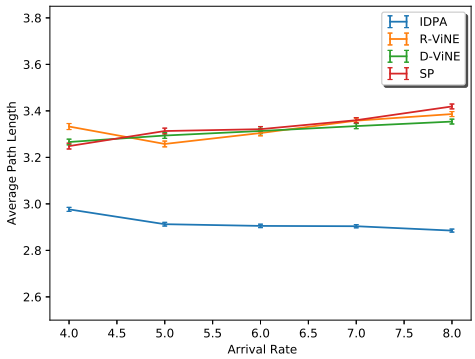


(b)

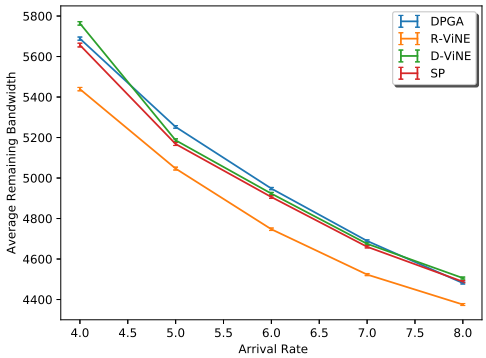
Figure 3.6: **(a)** Average cost **(b)** Average R/C ratio



(a) Average link utilization

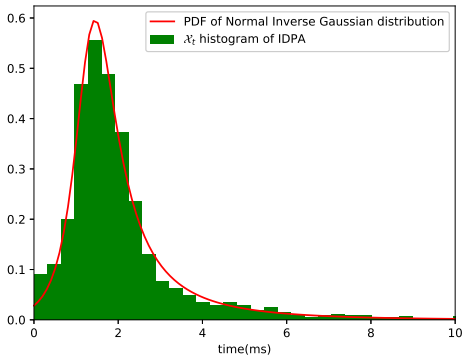


(b) Average path length

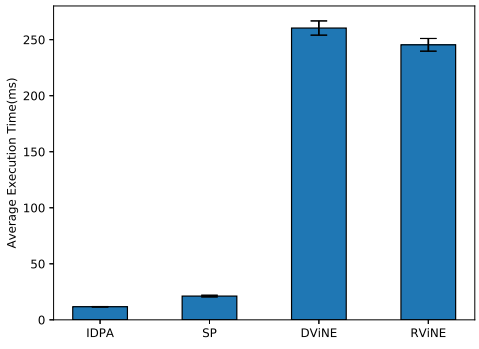


(c) Average remaining bandwidth

Figure 3.7: Average substrate link consumption



(a)



(b)

Figure 3.8: **(a)** Time distribution of $\mathcal{X}_t$ **(b)** Execution time over algorithms

3.4.10 Performance Results

We evaluate the efficiency of our proposed embedding solution for online VNE problem on several performance metrics as shown in Fig. 3.5, 3.6, 3.7 and 3.8. As delineated in Fig.3.5a and Fig. 3.6a, our proposed IDPA algorithm accepts more VN requests with less cost than all compared algorithms, which obviously results in higher revenue for our GA-based algorithm as acknowledged in Fig. 3.5b and 3.6b. The dominant achievements are because our proposed approach efficiently explores the searching space to discover more feasible solutions for evaluation. As our goal is to focus on the VLiM stage, the utilization of substrate resources towards physical links is substantially important. Fig. 3.7 illustrates that our embedding algorithm is highly efficient in approaching efficient solutions for VLiM phase. Furthermore, the average path length of our algorithm is much shorter than others leading to high remaining bandwidth that increases the possibility of accepting future VN requests. The superior performance is also from the effective FF that directs the GA algorithm to right track by minimizing both bandwidth usage and hop count factors. Thanks to parallel running, GA algorithm tends to examine various feasible solutions within a small time consumption. Besides our proposed algorithm outperforms the existing approaches in all performance metrics, The GA-based algorithm, IDPA, running on an appropriate distributed and parallel scheme is absolute faster than SP 44.87%, which has been known as the fastest and most popular unsplittable algorithm for VLiM. It is also faster than the best performance of our rivals R-ViNE $1,999\%$ as depicted in Fig. 3.8.

Our initially desired goals have been achieved when our proposal performs significantly better than all compared VNE algorithms in terms of performance and speed with very simple VNoM strategy. In next section, we enhance GA algorithm with multiple network attributes in the fitness function to achieve better performance than our previous algorithm IDPA. Details of this improvement can be found in our published paper [128].

3.5 Genetic Algorithm with An Enhance Fitness Function for VNE Problem

3.5.1 Introduction and Motivations

Network virtualization is an essential integral element of the future architecture networks (e.g. virtualized 5G networks, virtualized IoT networks), so the efficient and practical algorithms for solving online VNE problem are specially demanded. In [13], we proposed an intelligent parallel algorithm based on GA for online VN link embedding with both scalability and optimality taken into account. We found that our proposed algorithm ultimately outperformed the existing approaches in all performance matrices and IDPA is absolute faster than SP 44.87%, which is known as the fastest and most popular algorithm for VLiM. IDPA has been indeed proven to be efficient for solving online VNE problem with very simple VNoM strategy. However, the current fitness function of IDPA algorithm that combines between bandwidth and network hop-count factors is basically straightforward or uncomplicated. We wonder that a more complicated fitness function with additional substrate network features is able to drive the GA algorithm to better solutions for VNE problem compared to our previous algorithm IDPA? By such insatiable curiosity, we propose a GA-based distributed and parallel algorithm considering multiple objectives in fitness function for the link mapping stage. Our proposed GA algorithm, guided by a comprehensive fitness function, increases the opportunity to achieve efficient solutions and significantly reduces operation times due to its similar parallel deployment strategy in [13]. Expanding upon [13], we increase the complexity of the fitness function by introducing three more substrate resource attributes which are expected to improve performance results. We would also like to compare our proposed algorithm with another state-of-the-art VNE algorithm to concrete our achievements. Specifically, we would like to compete the new state-of-the-art algorithm in both performance and speed aspects, in addition to achieve better embedding performance against IDPA algorithm. We do not expect to defeat the speed of IDPA since both run on the same operation framework proposed in [13]. To the best of our knowledge, [128] is the first research paper that presents *attraction strength* as a factor in fitness function for VNE problems.

3.5.2 An Enhanced Fitness Function

GA is essentially a machine learning technique that attempts to address a specific problem from a population of candidate solutions. These candidates, known as chromosomes get evolved through selection, crossover and mutation operations relied on a grading mechanism, called the fitness function. FF determines the fitness of each chromosome, and can be used to select which chromosomes can be reproduced and survive in subsequent generations. The values of the fitness function impact the survival probability of a chromosome, with greater fitness values producing higher survival probability. This function can be applied to evaluate the quality of a virtual link embedding solution among several feasible ones. It can provide convincing scientific proof for ranking equivalent chromosomes in Genetic Algorithm operators.

In general, model objectives are converted into a form of fitness function that is typically utilized to guide the proposed network model to achieve the expected optimal solutions. The fitness function is correlated with design goals with multiple rigid constraints. It is additionally related to the multiple objectives to be optimized. However, FF must be computed quickly since a typical GA algorithm is iterated several times in order to achieve expected results. As a result, designing a FF that guides an efficiently model working while satisfying rapid execution requirements is challenging.

Similar to [13], we add hop-count to the fitness function as a substantial factor assuming that if the hop-count is minimized, the embedding cost will be minimized as well. In this section, we introduce three fitness factors to a new enhanced fitness function: embedding cost, distance factor, and attraction strength. The embedding cost factor adopts the link mapping solution that produces costs less than others while the distance factor presents the Euclidean Distance calculation (3.31) between consecutive nodes of a link to traverse a substrate path. Assuming that nodes n_1 and n_2 are determined by x and y coordinates (X_{n_1}, Y_{n_1}) and (X_{n_2}, Y_{n_2}), the Euclidean Distance of n_1 and n_2 can be calculated as follows:

$$\mathcal{D}_E = \sqrt{(X_{n_1} - X_{n_2})^2 + (Y_{n_1} - Y_{n_2})^2} \quad . \tag{3.31}$$

We recognized that the relationship among several constraints such as propagation delay, remaining link bandwidth and requested link bandwidth expressed a certain meaning. Thus, we integrated all these factors into a GA fitness function to determine

the best link embedding solution. Moreover, we proposed the attraction strength that indicated a consolidation of these aforementioned components to strengthen the fitness function.

Thus, the attraction strength of a link $l^s \in e^s : \forall e^s \in \mathcal{E}^s(\mathcal{A}_{\mathcal{L}}(l_i^v))$ can be calculated as follows:

$$\Theta(l^s, l^v) = K_c * \frac{R_L(l^s)}{b_r(l^v)} * \frac{\rho_r(l^v)}{\rho_d(l^s)} \quad , \tag{3.32}$$

where K_c is a constant, and $b_r(l^v)$ is the bandwidth requirement of the virtual link l^v. $\rho_d(l^s)$, and $\rho_r(l^v)$ are denoted as the propagation delay of the substrate link l^s and the delay request of the virtual link l^v, respectively.

These proposed factors not only improved the efficiency of the GA algorithm, but also compensated each other. The reason for this is that the distance factor would prefer solutions that have shorter geographical distance between substrate nodes, meaning that nodes which are close together tend to be selected. This enables node clustering to reduce resource fragmentation and effectively improve network utilization. Moreover, the attraction strength reinforces the fitness function by preferring the perfect match for the corresponding virtual link request. In a nutshell, the fitness function $\mathcal{F}(\mathcal{S}_z)$ is calculated as below:

$$\begin{aligned} \mathcal{F}(\mathcal{S}_z) = & \left(\frac{1}{\sum\limits_{n_i^v \in N_i^v} c(n_i^v) \; + \; \sum\limits_{l_i^v \in L_i^v} \sum\limits_{l^s \in L^s} f_{l^s}^{l_i^v}}\right) * w_c \; + \; \left(\frac{1}{\sum\limits_{l_i^v \in L_i^v} h(\mathcal{A}_{\mathcal{L}}(l_i^v))}\right) * w_h \\ & + \left(\sum_{l_i^v \in L_i^v} \sum_{l^s \in \mathcal{A}_{\mathcal{L}}(l_i^v)} \Theta(l^s, l^v)\right) * w_a \; + \; \left(\frac{1}{\sum\limits_{l_i^v \in L_i^v} \mathcal{D}_{E(\mathcal{A}_{\mathcal{L}}(l_i^v))}}\right) * w_s \quad , \end{aligned} \tag{3.33}$$

where, $\mathcal{S}_z$, h, Θ and $\mathcal{D}_E$ are a z^{th} feasible solution, hop-count, attraction strength and distance factor of the link mapping solution of l_i^v, respectively. w_c, w_h, w_a and w_s are weight parameters equivalent to cost, hop-count, attraction strength and distance factors. Assigning weights on different network attributes in fitness functions in GA algorithm is aimed at prioritizing important attributes that we would like to concentrate more on. For instance, hop-count has more weights than other factors since we would like to increase the possibility of accepting online VNRs in future. In this work, we set $w_c = w_s = 0.1$, $w_h = 0.5$ and $w_a = 0.3$.

Table 3.2: *Compared Algorithms*

Notation	Description
eIDPA	Intelligent Distributed and Parallel Algorithm with enhanced fitness function
IDPA	Intelligent Distributed and Parallel GA-based Algorithm [13]
ETAGNR	Exploiting Topology Attributes and Global Network Resources [68]
G-SP	Greedy Node Mapping with Shortest Path method for Link Mapping [12]
R-ViNE	Random Node Mapping with Shortest Path method for Link Mapping [12]
D-ViNE	Deterministic Node Mapping with Shortest Path method for Link Mapping [12]

3.5.3 Evaluation and Numerical Results

Compared Methods

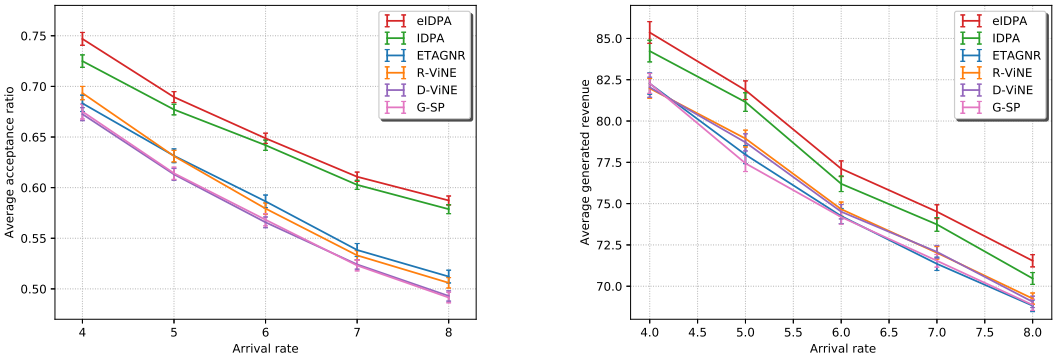
Similar to [13], we compare our improved GA-based algorithm with several competitors as listed in Table 3.2. Along with DViNE, RViNE and G-SP [12], we additionally compare our proposed algorithm with ETAGNR [68], a state-of-the-art VNE algorithm that employed a novel node ranking approach for node mapping, relying on multiple major topology attributes and global physical resources. Through a comprehensive comparison with a wide range of VNE approaches, both the superior performance and faster speed of our proposed algorithm can be conclusively confirmed.

Our simulation is also conducted on the same Ubuntu 19.10 (Eoan Ermine) 64-bit platform with 15.5 GiB memory and Intel Core i5-6200U CPU@2.30GHz×4.

Execution Time Analysis

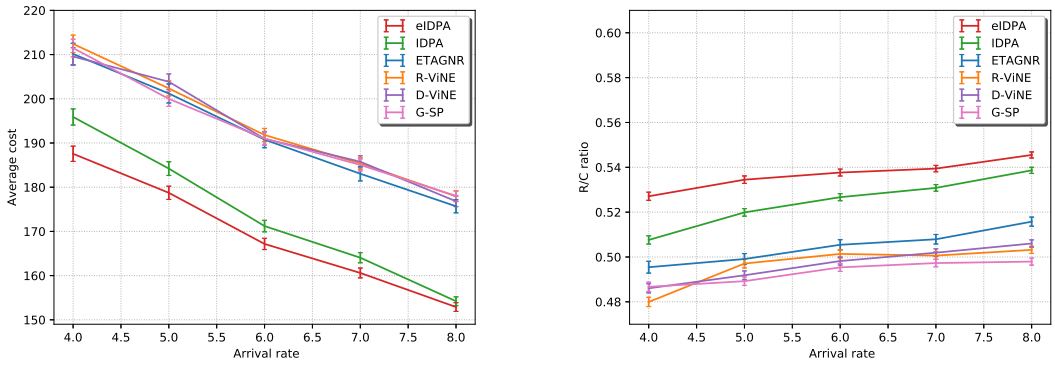
To determine execution time distribution, we collect the miscellaneous execution times of parallel procedures in the working nodes. As demonstrated in the $\mathcal{X}_t$ histogram of eIDPA in Fig. 3.13a, we can observe that $\mathcal{X}_t$ closely matches NIG. NIG is a normal variance mean mixture in which the mixing density follows the inverse-Gaussian

distribution. The square error measured is $6.0125e^{-3}$ while the sample mean and the standard derivation of $\mathcal{X}_t$ of eIDPA algorithm are 1.5295 and 0.856, respectively. Fig. 3.13b depicts the average execution time for embedding a VN on various compared algorithms where the parameter p of the eIDPA algorithm is configured with 16. In fact, we realize that with $p = 16$, the long-term performance of our proposed eIDPA algorithm tends to converge after extensive experiments. Fig. 3.13b depicts the average CPU execution time of a VNR on different algorithms. As illustrated, our proposed algorithm, together with the IDPA algorithm, significantly saves time compared with the competitors. eIDPA and IDPA are both even faster than G-SP which is widely accepted as the fastest solution for VN link mapping. There are very few research papers that compared the execution time of VNE with the shortest path method. This research is one of the exceptional research works proposing an innovative VNE algorithm that is significantly faster than the shortest path algorithm. It is not surprised that ETAGNR algorithm's execution time is the same with G-SP since ETAGNR utilizes the node-ranking approach for node mapping and the shortest path method for link mapping.



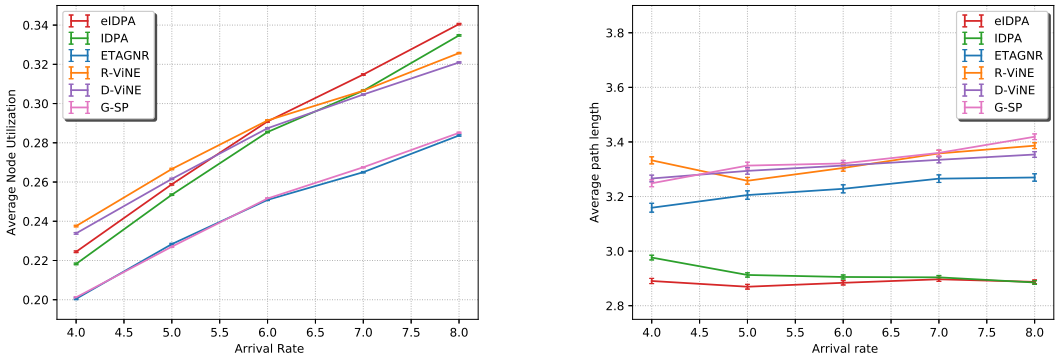
(a) **(b)**

Figure 3.9: **(a)** VNR acceptance ratio **(b)** Average generated revenue



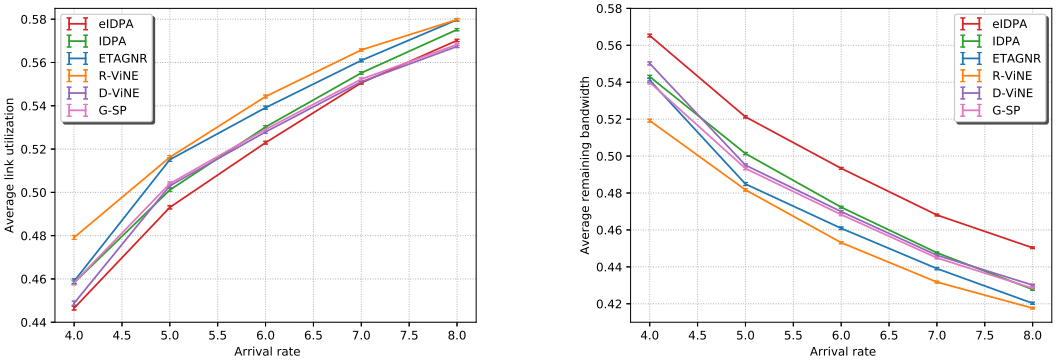
(a) **(b)**

Figure 3.10: **(a)** Average embedding cost **(b)** Revenue/cost ratio



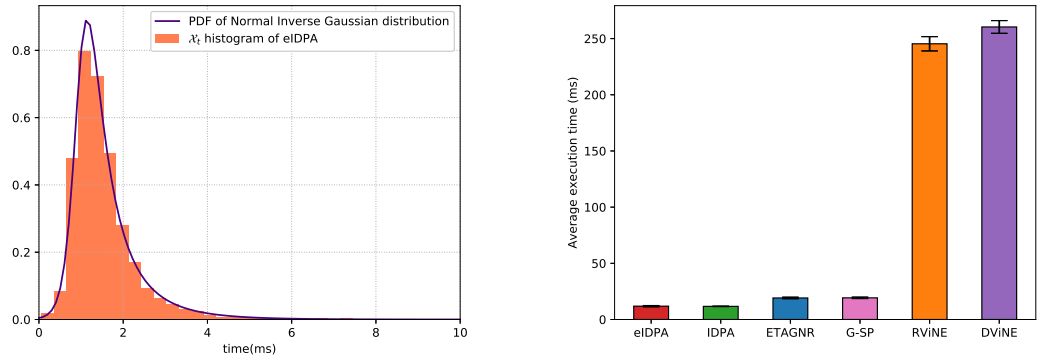
(a) **(b)**

Figure 3.11: **(a)** Average node utilization **(b)** Average path length



(a) **(b)**

Figure 3.12: **(a)** Average link utilization **(b)** Average remaining bandwidth



(a) **(b)**

Figure 3.13: **(a)** Time distribution of $\mathcal{X}_t$ **(b)** Execution time over algorithms

Evaluation Results

We extensively evaluate the efficiency of our proposed embedding solution for online VNE problem on several performance metrics as shown in Fig. 3.9, 3.10, 3.11, 3.12 and 3.13. As shown in Fig. 3.9, the eIDPA algorithm achieves significantly greater acceptance ratios with more accepted VNRs, and considerably lower costs than our rivals, including the IDPA algorithm presented in [13]. This results in higher revenues from our proposed algorithm. It is worth noting that we decided to employ the Greedy method for our node mapping due to its simplification, while D-ViNE and R-ViNE adopted a mathematical model to deal with the node mapping problem, and ETAGNR

deploys a state-of-the-art node ranking approach. Due to its novelty, ETAGNR performs better than other algorithms except IDGA and eIDGA. Not only does this result proves the effectiveness of our proposed GA algorithm but, more significantly, it indicates that link embedding plays a crucial role in addressing the VNE problem. In addition, the proposed algorithm efficiently performs when the substrate network is getting more congested as reflected by higher arrival rates. In Fig. 3.10, 3.11 and 3.12, eIDPA and IDPA [13] are apt to consume fewer network resources, thereby generating higher average revenue than other algorithms as depicted in Fig. 3.9b. The R/C ratio is determined by average revenue over cost as computed in Equation (3.11), so with the achieved performance results we can conclude that the R/C ratio of our algorithms (eIDPA and IDPA) is higher than that of the compared algorithms. Therefore, one more figure for R/C ratio is unnecessary. Moreover, our eIDPA algorithm is proven to be incredibly efficient due to reduced bandwidth use, as confirmed in Fig. 3.10b. Our proposed approach achieved these superior results by efficiently delving into the potential searching space in order to explore more feasible link mapping solutions. Furthermore, our enhanced fitness function properly guides the proposed GA-based algorithm by minimizing cost, propagation delay, hop count and, especially, distance strength factors. We believe that, together with other factors (e.g. cost), the distance strength element will efficiently decrease the network resource fragmentation problem by selecting link mapping solutions that have component substrate nodes that are geographically close. Thanks to our parallel operation scheme, the proposed algorithm eIDPA and even the IDPA algorithm seem to achieve a large number of feasible solutions for the link embedding problems in less time.

3.6 Rethinking Virtual Link Mapping in Network Virtualization

3.6.1 Introduction and Motivations

GA algorithm typically consists of four primary operators: initialization, selection, crossover and mutation. Crossover operator can be recognized as an exploitation phase in which the global optimum is positively expected to discover, so we argue that a creative crossover operator would improved the efficiency of our GA algorithm driven by an appropriate FF. Inspired from the DNA replication process in [131],

this mechanism allows cell division to occur. In primitive life forms, cell division vitally serves as an important means of reproduction, organism growth and cellular repair since the great number of cells can increase through cell division process. A parental cell can be split into two or more daughter cells by this process. They can inherit diverse combinations of partial or all DNA. In this research work, we redesign a novel crossover operator for the proposed GA algorithm. This new mechanism allows to proportionally exchange the random genes between the parental chromosomes to generate new offsprings.

As discussed in previous chapters, we assume that the role of link mapping stage in VNE is being underestimated, and the fastest speed of shortest path method can be vanquished by implementing an ingenious parallel operation scheme. Thus, we propose a novel intelligent GA-based orchestration algorithm for VLiM stage, running on a predefined number of independently distributed and parallel machines (e.g. virtual machines) to generate the feasible solutions denoted as chromosomes. To prove our hypothesis, we deploy a simple Greedy node mapping as the same with G-SP algorithm in [51] due to its simplicity. This selection can not only guarantee the rapid embedding speed, but also maximize the residual network resources leading more successfully allocated node mapping requests in future. Our proposed parallel GA scheme is also the same with Fig 3.2 in Chapter 4.

In this section, we propose a novel GA-based VNE algorithm that is relied on new design of the crossover operator for VLiM. The multi-constrained fitness function considering the embedding cost, hop-count and propagation delay has driven the proposal to an efficient VNE algorithm. To the best of our knowledge, this is the first research that applies an elastic crossover mechanism in GA algorithm to VNE problems. This paper is an extension of [13] towards the crossover operator and the improved fitness function, which not only achieves better performance than [13], but also outperforms state-of-the-art VNE algorithms.

3.6.2 Elastic Crossover for Genetic Algorithm

Crossover: this is literally considered as one of the most important step in GA algorithm, which primarily combines the parental chromosomes to produce new offsprings in next generations. We define $\mathcal{C}_s$ and $\mathcal{C}_r$ as the parental chromosomes with the corresponding indices s and r in the initial population. Different from typical GA crossover operator that normally employs one or more static random crossover points

to exchange the consecutive sequence of genes between two parental chromosomes, we proceed the crossover phase on selected chromosomes with different sequences of random numbers. Specifically, each related child has a different number of random genes to exchange with the others and vice versa. For example, Fig 3.14 illustrates our proposed crossover operator where the 1^{st} chromosome exchanges the 1^{st}, 2^{nd} and 5^{th} genes with the other. On the other hand, the 2^{nd} chromosome gets new 3^{rd} and 4^{th} genes from the first chromosome. This novel operation will provide an elastic crossover mechanism producing resilient offsprings, which is particularly desirable to improve the exploitation phase.

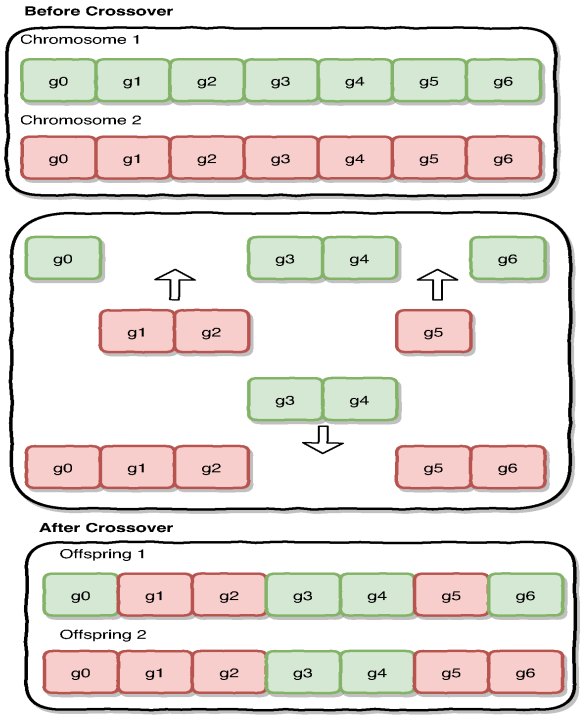


Figure 3.14: An example of the elastic crossover operator

Moreover, denote j^{c+n} as the random gene between any genes inbound the N length, and new descendant chromosomes are denoted as $\mathcal{C}_{(M+1)}$ and $\mathcal{C}_{(M+2)}$ respectively. The new offsprings are ultimately produced by changing random parent genes as depicted in (14b). Obviously, the quality of generated children may be worse than their ancestors or the duplication of solutions may happen at different parallel levels. The random genes of each parental chromosome j^{c+n} are randomly selected, and they can be even the first or last gene in the parents' chromosomes because the mated children are apparently based on distinct genes rather than a successive sequence of genes.

Furthermore, we might wonder how many genes are chosen to exchange because if all genes are selected, two parental chromosomes are just swapping all genes each other. In this case, our crossover stage is veritably dysfunctional. As a result, we define an elastic factor e^f which determines maximum number of genes that can be changed at each chromosome. It can be considered as a proportion of the N length. For instance, if $e^f = 0.75$ (75%) and $N = 7$, the number of genes that can be randomly chosen for mating is ranging from $[1 - 5]$

Fitness Function (FF): the fitness values of each solution determine which one will reproduce and remain "alive" in the next generation, relevant to the predefined objectives to be optimized in our proposed GA-based algorithm. As a result, this function is utilized to examine the quality of each VLiM solution among several feasible ones so that its values can provide a scientific proof for electing the corresponding solutions in GA stages. In details, we take the cost of embedding a VNR into consideration in this research, so solutions with less cost generated are definitely preferable. Moreover, we consider hop-count as an important factor into FF as it is substantially associated with bandwidth consumption. This means that less hop-count solution would consume less bandwidth, and then leaves more residual network bandwidth, increasing the possibility of the upcoming VNRs being accepted. The propagation delay of VLiM solutions is also estimated and added into FF as another constraint accompanying with the hop-count attribute in order to construct a multi-constrained fitness function. Fitness function $\mathcal{F}(\mathcal{S}_z)$ is eventually calculated as below:

$$\begin{aligned}\mathcal{F}(\mathcal{S}) = (\frac{1}{C(G_i^v)}) * w_c + (\frac{1}{\sum_{l_i^v \in L_i^v} h_{\mathcal{A}_\mathcal{L}(l_i^v)}}) * w_h \\ +(\frac{1}{\sum_{l_i^v \in L_i^v} d_\mathbb{P}(\mathcal{A}_\mathcal{L}(l_i^v))}) * w_p\end{aligned} \tag{3.34}$$

where, $\mathcal{S}$, h and $d_\mathbb{P}$ are a feasible solution, hop-count and propagation delay of the link mapping solution of l_i^v respectively. w_c, w_h, and w_p are weight parameters equivalent to cost, hop-count and propagation delay factors.

Table 3.3: *Compared Algorithms*

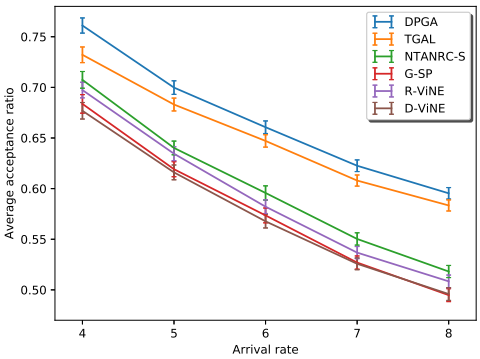
Notation	Description
DPGA	Novel Distributed and Parallel GA-based Algorithm proposed in this section
TGAL	Typical GA-based Algorithm for Link Mapping [13]
NTANRC-S	Network Topology Attribute and Network Resource-considered algorithm (Stable) [69]
G-SP	Greedy Node Mapping with Shortest Path Based Link Mapping [51]
R-ViNE	Random Node Mapping with Shortest Path Based Link Mapping [51]
D-ViNE	Deterministic Node Mapping with Shortest Path Based Link Mapping [51]

3.6.3 Performance Evaluation

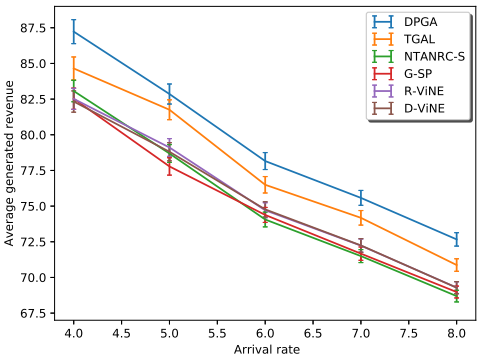
Comparison Methods

In our performance evaluation, we compare the proposed GA-based algorithm with the selected competitors as listed in Table 3.3. In order to prove the effectiveness of our proposed algorithm, the state-of-the-art VNE algorithm, namely NTANRC-S, based on node ranking method is selected for comparison in addition with algorithms in [12]. The reasons we choose [12] are explained in previous chapters. Additionally, NTANRC-S algorithm is preferred because it shows better performance than NTANRC-D in [69]. It is also the most recent algorithm represented for node ranking category as investigated in Section 2.3.2. Moreover, we select TGAL [13] since it is one of the first algorithms deliberately designed for link mapping stage, and it is primarily based on the typical GA-based algorithm running in parallel. By comparing with TGAL, we validate the improvement of our proposed GA-based algorithm in this chapter. Similar to [13] we also choose the node embedding algorithm the same as G-SP for a fair comparison. It is expected that our proposed VNE algorithm not only outperforms NTANRC-S algorithm in performance, but also still achieves a faster speed than G-SP due to an enabled parallel operation design.

Our simulations also run on the same Ubuntu 19.10 64-bit platform with 15.5GiB memory and Intel® Core™i5-6200U CPU@ 2.30GHz×4.

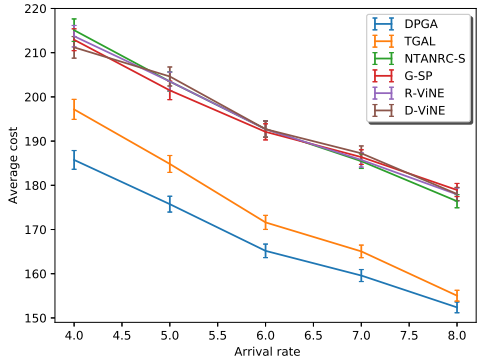


(a)

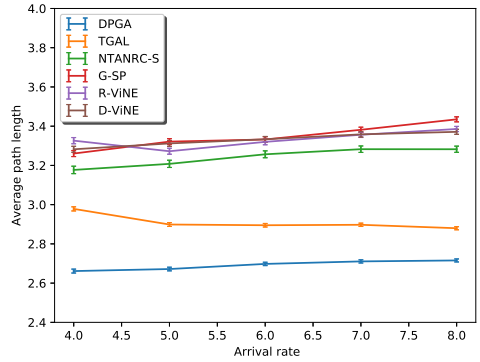


(b)

Figure 3.15: **(a)** VNR Acceptance ratio **(b)** Average generated revenue

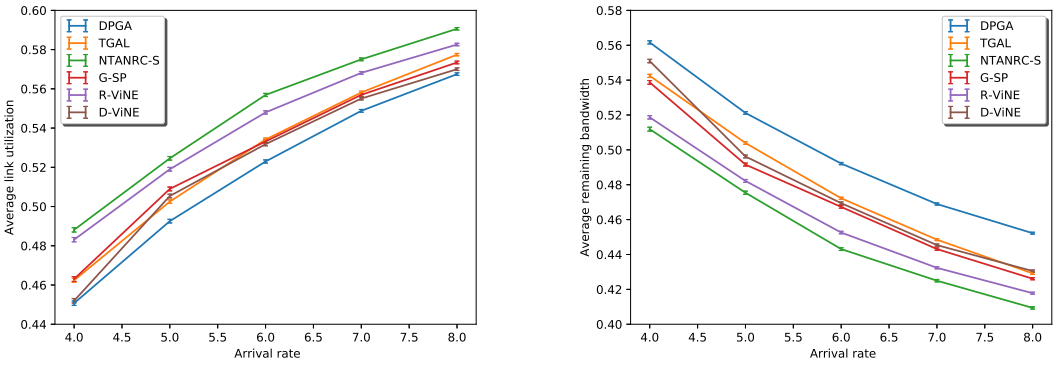


(a)



(b)

Figure 3.16: **(a)** Average cost **(b)** Average path length



(a) **(b)**

Figure 3.17: **(a)** Average link utilization **(b)** Average remaining bandwidth

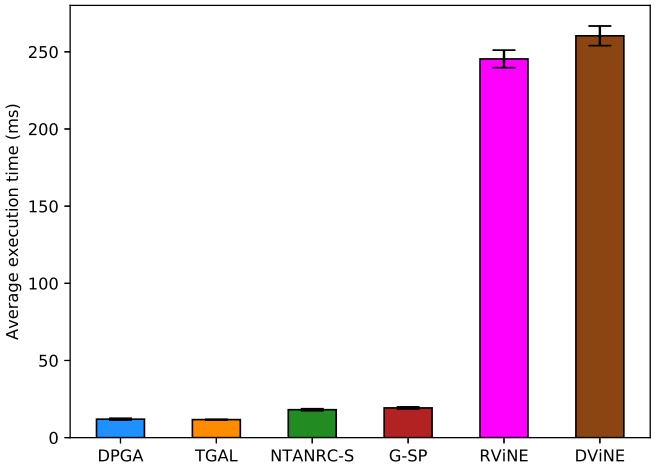


Figure 3.18: Execution time over different algorithms

Execution Time Analysis

We also collect miscellaneous operation time of the parallel procedures in several working nodes. Similar to IDPA or eIDPS in [13] and [128] respectively, $\mathcal{X}_t$ of DPGA greatly fits NIG defining as a normal variance-mean mixture in which the mixing density follows the inverse Gaussian distribution. The sum of square error measured is $2.28e^{-3}$, whilst the sample mean as well as the standard derivation of $\mathcal{X}_t$ of DPGA are measured as 1.801ms and 1.048 respectively. Additionally, the average total execution

time of a VNR on different compared algorithms where the parallel level p of DPGA algorithm is configured with 16 is depicted in Fig.3.18. The overall performance of DPGA after numerous experiments is converged with $p = 16$. As observed, our proposed algorithm remarkably save time compared with the competitors, and it is even faster than G-SP as expected. We propose IDPA, eIDPA and now DPGA as strawman proposals for how metaheuristic approaches can effectively compete the G-SP in term of speed.

Evaluation Results

As shown in Fig. 3.15 and Fig. 3.16, DPGA algorithm achieves greater acceptance ratios by accepting more VNRs with significantly less total costs than all competitors as depicted in Fig. 3.16a, which obviously leads to much higher revenue for our proposed algorithm (Fig. 3.15b). Specifically, DPGA performs better than R-ViNE - the best performance in [12], NTANRC-S and TGAL for approximately 9.14% (17.1%), 7.58% (14.91%) and 3.95% (2.1%) at the acceptance ratios of 4 (8) respectively as represented in Fig. 3.15a. In this research, we substantially concentrate on the VLiM problem, so the average remaining bandwidth of all compared algorithms is selected and illustrated in Fig. 3.17b. When more VNRs arrive, the link utilization of compared algorithms unsurprisingly increase. However, our link mapping result achieves superior performance since it consumes less bandwidth to embed the link mapping requests, which is confirmed by Fig. 3.17 and 3.16b. More remaining bandwidth means higher probability of accepting new VNRs. DPGA and TGAL obtain much better average path length metric compared to other algorithms as shown in Fig. 3.16b, which contributes to lower costs and better remaining bandwidth results. This is because we consider the hop-count factor in FF to determine the VNE link embedding solution.

In addition, the average CPU execution time of DPGA embedding one VNR is almost the same as TGAL, but our proposed algorithm achieves an absolute faster speed than G-SP for more than 43% as illustrated in Fig. 3.18 as expected. It is observed that NTANRC-S nearly performs the same execution time in comparison with G-SP as it also deploys the node-ranking heuristic algorithm for VNoM and the shortest path method for VLiM.

The remarkable performance is indeed gained because our proposed GA-based approach explores the searching space efficiently in order to acquire more feasible link embedding solutions due to a novel crossover operator. In addition, our fitness function

guides the GA-based algorithm on the-right-track by minimizing the total embedding cost, propagation delay and hop-count factors. Furthermore, our novel GA algorithm tends to evaluate various feasible solutions in a greatly less time consumption due to a proper parallel operation paradigm as depicted in Fig. 3.18. Accordingly, the time complexity of our GA-based algorithm is rapidly reduced to logarithmic $O(log(p))$ as detailed in Section 3.3.

3.7 Efficient VNE with Node Ranking and Intelligent Link Mapping

3.7.1 Introduction and Motivations

We have recognised that virtual link embedding also plays an important role in approaching efficient solutions for embedding VN requests. In Chapter 3, we only focus on VLiM stage while leaving the node mapping stage for very simple node mapping that was proposed in [43]. Motivations and reasons for this choice are clarified in previous Chapter so we would not like to explain here again. However, this selection would raise another concern. How good or bad achieved if the VNoM is "upgraded" to a more complicated algorithm. How much gains InPs can obtain with that choice. From this perspective, we advise Network topology attributes and network resource-considered (NTANRC) algorithm, a VNoM mechanism, that considers essential network features and global network resources for ranking both substrate and virtual nodes prior to embedding each given VNR. Accordingly, NTANRC combined with a distributed and parallel Genetic Algorithm for VLiM, namely NTANRC-GA, to solve online VNE problem is investigated in this chapter. NTANRC algorithm and the parallel GA-based algorithm are reverse compliments of each other to achieve an efficient VNE solution.

To the best of our knowledge, this is the first paper that efficiently deploys a node-ranking approach incorporated with a complicated link mapping solution to solve VNE problems. It is expected that our proposed VNE solution improves the acceptance ratios of VNRs, revenue to cost ratios and link resource utilization compared to NTANRC [69], IDPA [13] and three VINE algorithms [12]. Furthermore, we hope that the proposed solution will keep rapid embedding speed like previous proposals by deploying the distributed and parallel operation scheme in Chapter 2.

3.7.2 Node Ranking for VNoM and Distributed Parallel GA-based Algorithm for VLiM

NTANRC-GA implements NTANRC algorithm [69] for addressing VNVoM while deploying the distributed and parallel GA algorithm [13] for link mapping. It means that the output of NTANRC algorithm becomes the input for VLiM. NTANRC algorithm efficiently seeks the potential embedding nodes for VNRs. Besides, the novel GA-based algorithm that is based on a distributed and parallel scheme efficiently reduces the embedding cost, execution time, and improves link utilization.

NTANRC Algorithm for Node Mapping

NTANRC collects five network topology attributes for its node-ranking adoption including node degree, node strength, distance, farness and closeness and finally link interference. As shown in [69], there are two sub-approaches (S: stable and D: direct), but we prefer S-approach due to its better performance.

Node Degree: Let $\tilde{\mathcal{T}}_d(.)$ denote the function that counts the total number of adjacent links of node n^s, degree of a substrate node is determined as follows:

$$\Gamma(n^s) = \tilde{\mathcal{T}}_d(n^s) \tag{3.35}$$

Node Strength is defined by the function $\tilde{\mathcal{T}}_s(.)$ that counts the sum of all adjacent link bandwidth of the substrate node n^s.

$$\Theta(n^s) = \tilde{\mathcal{T}}_s(n^s) \tag{3.36}$$

Farness and Closeness of A Node: Farness of a node n^s is the sum of its shortest distances to all the other possible nodes whereas Closeness is reciprocal of the Farness as defined in (3.37) and (3.38).

$$F(n^s) = \sum_{m^s \in N^s} \mathcal{D}(n^s, m^s) \tag{3.37}$$

$$Z(n^s) = \frac{1}{F(n^s)} \tag{3.38}$$

Link Interference: Interference of a physical link $l^s_{(n^s,m^s)} \in L^s$ between node n^s and node m^s describes its contribution to the network connectivity, relying on the

idea that traffic routed in a path can be minimized traffic interference in the future.

$$I(l^s_{(n^s,m^s)}) = \frac{F(n^s)}{\Gamma(n^s)} + \frac{F(m^s)}{\Gamma(m^s)} \tag{3.39}$$

In node-ranking approach, topology attributes and the global network resources are simultaneously quantified in which the node capacity, node location, link bandwidth and link propagation delay are considered as global resource parameters. Accordingly, interaction between two nodes n^s and m^s denoted as $\Phi_{(n^s,m^s)}$ is defined as follows:

$$\Phi_{(n^s,m^s)} = \epsilon_I \frac{\mathcal{B}(n^s)\mathcal{B}(m^s)}{\mathcal{D}(n^s,m^s)^2 d_{\mathbb{P}}(n^s,m^s)^2} \tag{3.40}$$

$$\Phi_{(n^s)} = \sum_{n^s \neq m^s, m^s \in N^s} \Phi_{(n^s,m^s)} \tag{3.41}$$

where ϵ_I is a constant, and Eq. 3.41 expresses the interaction of a node n^s with remaining nodes in the entire substrate network. In simulation, Eq. 3.40 is applied for each pair of the corresponding node with all other nodes in SN and then Eq. 3.41 sums all calculated values to provide the eventual interaction values. Moreover, $\mathcal{B}$ denotes the resource block of a substrate node that can be defined as follows:

$$\mathcal{B}(n^s) = c(n^s)\Theta(n^s) \sum_{m^s \neq n^s, m^s \in N^s} I(l^s_{(n^s,m^s)}) \tag{3.42}$$

Normalized $\mathcal{B}(n^s)$ and $\Phi_{(n^s)}$ towards substrate node n^s are formulated as below:

$$\bar{\mathcal{B}}(n^s) = \frac{\mathcal{B}(n^s)}{\sqrt{\sum_{n^s \in G^s} \mathcal{B}(n^s)^2}} \tag{3.43}$$

$$\bar{\Phi}_{(n^s)} = \frac{\Phi_{(n^s)}}{\sqrt{\sum_{n^s \in G^s} \Phi_{(n^s)}{}^2}} \tag{3.44}$$

By calculating all percentage values $\bar{\Phi}_{(n^s)}$ of the substrate network G^s, an initial node-ranking vector T_0 has been made, where $T_0 = (\bar{\Phi}_{(n^0)}, \bar{\Phi}_{(n^1)} \cdots \bar{\Phi}_{(n^{|N|})})^T$. For each node $n^s \in N^s$, its node-ranking value can be calculated as follows:

$$\nu_{n^s} = (1-\rho)\bar{\mathcal{B}}(n^s) + \rho \sum_{n \neq m, m \in N(n^s)} \Phi_{(n^s)} \nu_{m^s} \tag{3.45}$$

where ρ is the damping factor and $N(n^s)$ is the set of all nodes that have a path with the node n^s in the substrate network. As a result, a vector $\mathcal{V}$ formed by the node-ranking values of all substrate nodes can be shown as below:

$$\mathcal{V}_{n^s} = (1-\rho)\bar{\mathcal{B}}(n^s) + \rho\mathcal{M}\mathcal{V}_{n^s} \tag{3.46}$$

where $\mathcal{V}_{n^s} = (\nu_{n^0}, \nu_{n^1}, \cdots, \nu_{n^{|N|}})^T$, and $\bar{\mathcal{B}}(n^s) = (\bar{b}(n^0), \bar{b}(n^1), \cdots, \bar{b}(n^{|N|}))^T$. $\mathcal{M}$ is the transition matrix ($|N| \times |N|$) where each element is calculated based on (3.43) and (3.44). Details on a stable node-ranking approach are described in Algorithm 2 that is utilized to calculate the node-ranking values. NTANRC requires that all nodes including substrate and virtual nodes should be ranked and sorted following a decreasing order of the node-ranking values. Similar to substrate nodes, the node ranking values for virtual nodes are computed in the same procedures. When a VNR arrives, virtual nodes are calculated node-ranking values and then the virtual node with highest node-ranking value is first mapped to the substrate node that achieved highest node-ranking value, meeting node location and capacity constraints. In a VNR, virtual nodes cannot share the same substrate node. The mapping is repeated until all virtual nodes are successfully mapped to corresponding substrate nodes. The output of NTANRC algorithm as node mapping results is then utilized as the input to the parallel GA-based algorithm [13] to deal with VLiM.

Algorithm 2 Stable Node-Ranking Approach

Input: a network G = (N,L), δ: small positive number
Output: a ranking vector $\mathcal{V}$ corresponding to a given G
Calculate matrix $\mathcal{M}$ and initial vector $\mathcal{V}_0(T_0)$
Define iteration number k and variable w.
while $w \geq \delta$ **do**
    $\mathcal{V}_{k+1}$ = (1- ρ)$\bar{\mathcal{B}}(n^s)$ + ρ $\mathcal{M}$ $\mathcal{V}_k$;
    w = $\|\mathcal{V}_{k+1} - \mathcal{V}_k\|$;
    k = k +1;
$\mathcal{V}_k = \mathcal{V}_{k+1}$

Parallel GA-based Algorithm for Link Mapping

Distributed and Parallel computing has considered as an efficient mechanism to address large and complex problems with lower costs and less time consumption by concurrency support. GA is an appealing AI approach for solving constrained

or unconstrained optimization problems. A conventional GA usually includes four main operators: initialization, selection, crossover and mutation. An intelligent GA-based orchestration algorithm for VLiM stage not only approached an efficient VNE solution, but also accelerated the embedding speed due to a distributed and parallel paradigm [13]. This virtual link embedding solution is the same with IDPA in [13] that was clearly described in Chapter 4, so we do not elaborate it again. The reason for the IDPA choice is that first IDPA is one of the first VNE solution relied on GA algorithm that merely focuses on VLiM; second, it deploys very simple strategy for embedding VN link requests compared to eIDPA or DPGA in Chapter 4. From that, we can basically quantify how much gains from the advanced node mapping.

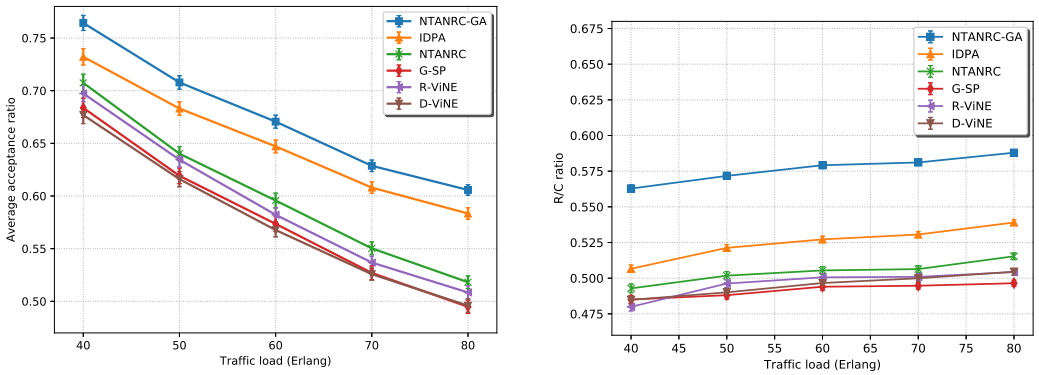
3.7.3 Evaluation and Numerical Results

We compare our proposed NTANRC-GA algorithm with several competitors including IDPA [13], NTANRC-S [69], D-ViNE, R-ViNE, and G-SP [12]. These algorithms are evaluated in various performance metrics including acceptance ratio, average revenue to cost ratio, remaining bandwidth and execution time.

Simulation Setup

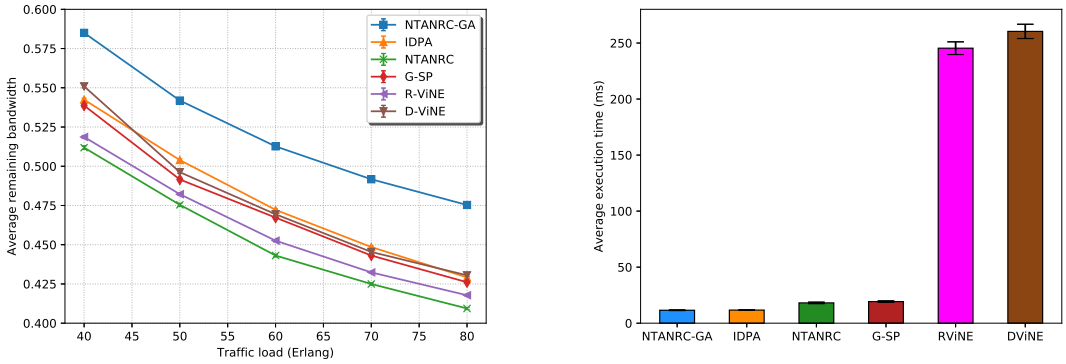
We also deploy a discrete-event simulator to assess our proposed VNE solution. Most of simulation parameters such as network topology generator, substrate or virtual resource capacity, arrival distribution, life time are similar to [13] and [12]. However, the load of VN requests is quantified by $\frac{\lambda}{\mu}$ Erlangs. The error bars were very small due to a large number of samples used, which proved that our simulation results were obviously reliable. For better presentation, we plotted figures with different colors and markers.

Simulation results are preferably shown in Figure 3.19 and 3.20. NTANRC-GA achieved highest acceptance ratio by accepting more VNRs with significantly less costs than all rivals, which resulted in much higher average revenue to cost ratios as depicted in Fig 3.19b. Achieving higher both acceptance and revenue to cost ratios simultaneously is desirable for any VNE algorithm. Our proposed solution proved that it could be feasible. For example, NTANRC-GA improved the acceptance ratios of IDPA as well as NTANRC, and was better than R-ViNE (the best performance



(a) **(b)**

Figure 3.19: **(a)** VNR Acceptance ratio **(b)** Revenue/Cost ratio



(a) **(b)**

Figure 3.20: **(a)** Average remaining bandwidth **(b)** Execution time over algorithms

algorithm in [12]) up to 2.24%, 8.76% and 9.73% at 80 Erlang (Fig. 3.19a) respectively. Our proposed VNE solution gained 9.07%, 14.08% and 16.60% better revenue to cost ratios than those of aforementioned algorithms at the same traffic load as illustrated in Fig 3.19b. With the intelligent link mapping, our VNE solution consumed less bandwidth to embed given link mapping requests due to the efficient FF as shown in Fig. 3.20a.

Moreover, we compared the execution time for embedding a VNR among algorithms where those of NTANRC-GA and IDPA algorithms are almost similar because they use the same link mapping mechanism. As depicted in Fig .3.20b , NTANRC-GA is

36.26% and 40.11% absolute faster than NTANRC and G-SP algorithms, respectively. The reason is that both NTANRC-GA and IDPA deploy the distributed and parallel GA-based algorithm for link mapping stage instead of the SP method used in NTANRC and G-SP, which was proved to be considerably faster than SP in [13]. The remarkable performance came from the effective node-ranking method, that simultaneously took network topology attributes and global network resources into account, and the intelligent GA-based approach exploring the search space efficiently to achieve more feasible link mapping solutions with lower time consumption due to a proper parallel operation paradigm. With regard to time complexity and convergence, interested readers may refer to Section 3.3 for further theoretical analysis.

3.8 Chapter Summary

Network virtualization is indisputably a primary component of future network architectures (e.g. virtualized fifth generation networks, smart IoT), so identifying an efficient VNE algorithm is crucial. In this chapter, scalability and optimality are simultaneously taken into account to solve online VNE problem. A generic distributed and parallel framework for virtual resource allocation problem and its running time model are also introduced.

We then propose an intelligent parallel algorithm based on GA for online VN link embedding, namely IDPA. Our proposed algorithm ultimately outperforms the existing approaches in all performance matrices. Moreover, IDPA is absolute faster than SP 44.87%, which is known as the fastest and most popular algorithm for VLiM. It is also faster than the best performance of our rivals R-ViNE $1,999\%$. An extension of IDPA, namely eIDPA algorithm which is strengthened by an enhanced multiple-constraint fitness function is also introduced. Our proposed algorithm, driven by a novel fitness function, outperforms its competitors in all evaluation matrices, including performance and time efficiency. eIDPA is also 44.01% faster than G-SP and achieves better acceptance ratios than the best performance in [12] (D-ViNE) and the state-of-the-art VNE embedding algorithm in [68] (ETAGNR) (8.13% and 7.52%, respectively) with an arrival rate of 8.

Furthermore, we redesign the crossover operator and present a multi-constrained fitness function considering multiple network attributes guiding GA algorithm towards an efficient VNE solution. Our proposed GA-based algorithm eventually outperform

state-of-the-art algorithms in all evaluation matrices including performance and time efficiency. In essence, DPGA remarkably achieves an absolute faster speed than G-SP by 43.5%, which is widely accepted as the fastest and most popular VLiM algorithm. Moreover, it gains better performance than TGAL in [13] in all comparison metrics.

We also provide a mathematical analysis of the operation time of the distributed and parallel model. Our statistics confirm that time complexity is rapidly reduced to logarithmic $O(log(p))$ in our parallel operation scheme, compared to as the sequential one.

Until now, we have proposed metaheuristic approaches implemented in an appropriate operation framework including IDPA, eIDPA and now DPGA as strawman proposals to effectively compete the G-SP in term of speed and the state-of-the-art VNE algorithms regarding performance. However, these proposed algorithms are just utilizing simple VNoM, specifically Greedy method, in order to prove the efficiency of our proposed distributed and parallel GA-based algorithms.

Therefore, we introduce the node-ranking approach that is based on network topology attributes and global network resources for VNoM stage and the intelligent parallel GA-based algorithm for link mapping stage to solve online VNE problem. Our proposed NTANRC-GA algorithm outperforms previous VNE algorithms in all matrices towards performance and running time. We can conclude that NTANRC and intelligent GA-based algorithms are reverse compliments of each other to achieve an efficient VNE solution.

In next chapter, we will investigate a joint node and link mapping approach which can solve a conflict between uncoordinated node and link mapping stages. A novel conciliation algorithm is proposed to handle a possible set of infeasible link mappings caused by inappropriate node mapping. It is aimed at minimizing the number of remapped nodes and links.

Chapter 4

Joint Node and Link Algorithms for Embedding VNs with Conciliation Strategy

4.1 Introduction

Til now, we have proved that virtual link embedding plays a crucial role in achieving efficient solutions for embedding online VNRs in previous chapter. Like most of VNE approaches that have focused on mapping VNs in separate stages with scalable heuristic algorithms. Chapter 3 proposes several efficient VNE algorithms for VLiM stage where VNoM is assigned to either the simple greedy mapping or node ranking method. The decoupling between VNoM and VLiM stage simplifies the complexity of algorithm deployment, but this common approach most likely scarifies some degrees of optimality. Consequently, the lack of a coordination results in low acceptance ratio as well as low average revenue. For example, although VNoM stage may come up with good solutions for mapping virtual nodes in a given VNR, one of virtual links that is failed to map onto the SN due to network congestion causes the rejection of the whole given VNR despite of successfully already-mapped virtual links as well as previous node mappings. As stated in previous chapters, the most common failures of VNs mapping are invariably emanate from the infeasible link mapping. This situation is understandable since the input of VLiM that is the mapping solution of all virtual nodes is fixed, and VLiM has no chance to remap the virtual nodes in previous VNoM stage. In this chapter, we present a GA-based algorithm to solve VNE problem by jointly mapping virtual nodes and virtual links. GA attempts to map all virtual nodes

in VNE at first and then corresponding virtual links will be sequentially embedded based on different path searching methods. To handle possible failures of mapping virtual links due to network congestion, current node mappings of these associated virtual links need to be remapped with minimal node and link mapping changed. As result, we recommend an efficient conciliation algorithm to address this problem. We also implement our proposed algorithm in the distributed and parallel scheme to reduce the execution time, and compare the result with the one deployed in sequential scheme.

To the best of our knowledge, this is the first work that introduces a joint node and link mapping approach assisted by a novel conciliation strategy in GA for solving online VNE problems efficiently. It is expected that our proposed VNE solution outperforms all state-of-the-art VNE algorithms and will become a challenge for any future VNE algorithm.

4.2 Our Proposed Joint Node and Link Mapping with Conciliation Strategy

Our proposed parallel GA scheme is presented in Fig 4.1. Each working machine

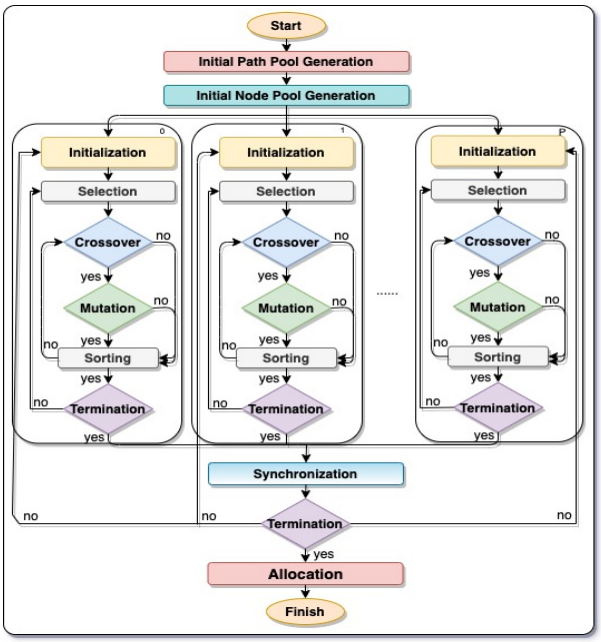


Figure 4.1: *Parallel operation scheme*

independently runs GA algorithm with a pre-defined number of iterations to explore feasible solutions for a VNR. The best-matching outcome is selected among these parallel machines. When node mappings are changed, the link mappings altered accordingly.

4.2.1 Node Mapping Algorithm

In our proposed joint node-link mapping algorithms where the GA algorithm is the core, the node mapping of a given VNR simply works in a heuristic way. Population in GA's operations consisting of several chromosomes is generated in random manner. A VNR includes a set of virtual nodes connected with several virtual links. Accordingly, there are two separate sets of solutions for virtual nodes and virtual links. The later solutions are dependent on the former ones, so if the node mappings are changed, the link solutions are altered correspondingly. In details, our proposed GA algorithm defines a specific mapping of a virtual node as a gene. Thus, a chromosome $\mathcal{C}_f$ represents a specific mapping of all the virtual nodes of a VNR. Each gene Φ_f^j is associated with a mapping solution of a virtual node, where f and j indicate the f^{th} chromosome and j^{th} virtual node in a VNR respectively. We equally consider all virtual node mappings. Therefore, their order in a chromosome can be assigned in an arbitrary manner. Once assigned, all the chromosomes will follow the same order. To enhance the capability and reduce execution time, we initially generate a feasible node pool where all eligible substrate nodes meeting the least virtual node requirements of the VNR are collected. Each virtual node in a VNR is consecutively mapped by a substrate node which is randomly selected in the initial node pool, that must meet resource requirements (Eq. (3.1)-(3.4)) to become a feasible node mapping. In fact, we assume mapping virtual nodes is carried out in random to prevent the possible premature convergence problem of GA algorithm.

4.2.2 Link Mapping Algorithms

When the substrate nodes are identified, we then embed the virtual links in VNR onto substrate paths where each path includes several physical links. Seeking an optimal mapping for a virtual link mapped to a single physical path with already-completed node mapping can be reduced to unsplittable flow problem which has been widely recognized as $\mathcal{NP}$-hard [69].

Moreover, we argue that substrate paths for each pair of physical nodes in a SN can be determined in advance due to the fact that topology of a SN is most likely static. Consequently, the path database for VLiM, called initial path pool generation in Fig. 4.1, can be constructed completely prior to the arrival of online VNRs. The shortest path method (e.g., Dijkstra's algorithm) is used to build this database with respect to the hop-count factor to enhance possible fragmentation problem. Each virtual link is embedded onto a single substrate path between the substrate nodes hosting corresponding virtual nodes of the given virtual link. In this section, we propose four virtual link mappings with different searching mechanisms.

Random Path Searching Algorithm (*GA-RAN*)

Towards a virtual link request, the path searching algorithm randomly selects a possible path in the path database that has been already built in advance as shown in Fig. 4.1, based on the information of node mappings. This path is required to pass a validity check meeting the resource constraints Eq. (3.6)-(3.8) to become a feasible path. If the path fails to pass this checking process, the algorithm attempts to find another one in the database until at least one feasible path achieved. If there is no feasible path found in the path database, the algorithm returns failed virtual link request. Accordingly, the previous node mappings should be reconsidered. Each virtual link in the given VNR is sequentially searching for a feasible substrate path in that way until all virtual links are processed.

Sequential Single Path Searching Algorithm (*GA-SEQ*)

Instead of searching the path in random, when a virtual link request arrives, the sequential path searching mechanism finds a potential path in the path database following the node mappings given. As a result, the algorithm starts searching from the first path in the database until the last one. Whenever there is a substrate path that passes the validity check, it becomes feasible and then the searching algorithm will stop there. As said, the erected path database is relied on the hop-count factor, so the first path in database has the lowest hop-count feature. In contrast with the random manner, the sequential path searching goes through the path database by sequentially searching from the first to the last path until a feasible one found. After the whole path database is inspected, if there is no feasible path achieved, the

algorithm returns failed for the corresponding link mapping similar to the random path searching. All virtual links in the VNR are handled in the same way. The link mappings that are specified as "failed" will be then reprocessed in the conciliation algorithm. Due to the simple and straightforward feature of the sequential single path searching mechanism, we expect that its execution time of this algorithm is better than those of other proposed algorithms.

Shortest Path Algorithm (*GA-STP*)

Different from the aforementioned path searching algorithms, we deploy k-shortest path algorithm as an approximation approach so as to minimize bandwidth consumption of a successfully mapped virtual network. Based on the information of node mappings, we map each virtual link of the VNR to a single physical path. The shortest path algorithm searches k-shortest paths by increasing the value of k until a feasible path that has sufficient bandwidth to embed the corresponding virtual link is found. The time complexity of the algorithm increases with the value of k. For an efficiency of computation and network resource usage as well as reducing link delay, the value of k should be kept minimal, so we choose $k = 1$ in this paper. In case, there is no feasible path found, the according virtual link is marked as failed. The process is stopped when the mapping of all virtual links is established and some of them might be "failed". Instead of increasing k to seek for the "second shortest", remapping virtual nodes of the corresponding failed virtual links is implemented to find the best link mappings. With these objectives, we expect that our proposed joint node-link mapping assisted by the shortest path mechanism is able to achieve the best performance compared to its counterparts, especially the average link delay metric. A desirable outcome towards the delay metric is beneficial for the virtual network requests that strictly require sensitive delay requirements (e.g., IoT applications).

Path Ranking Algorithm (*GAPK*)

We argue that substrate paths for each pair of source and destination nodes in a SN can be determined in advance due to the fact that topology of a SN is basically static. Consequently, path database for VLiM can be constructed completely prior to the arrival of online VNRs. The shortest path method (e.g., Dijkstra's algorithm) is used to build this database, called initial path pool generation in Fig. 4.1. When there is a virtual link request, the path ranking algorithm sequentially finds a predefined

number of feasible paths in database for each source-destination pair (e.g., r = 10) based on the node mapping information of such request given. This number is large enough to guarantee a good mapping for any link request and it can be adjusted to compromise a trade-off between expected performance and execution time. These paths can then be ranked by their fitness values (Eq. 3.33) to select the best mapping solution for the virtual link request.

4.2.3 Conciliation Construction

In GA's operations, population consisting of several chromosomes is generated in random manner. A VNR includes a set of virtual nodes connected with several virtual links. Accordingly, there are two separate sets of solutions for virtual nodes and virtual links. The later solutions are dependent on the former ones, so if the node mappings are changed, the link solutions are altered correspondingly. Hence, our proposed GA algorithm defines a specific mapping of a virtual node as a gene. A chromosome $\mathcal{C}_f$ is a specific mapping of all the virtual nodes of a VNR. Each gene g_f^j is associated with a mapping solution of a virtual node, where f and j indicate the chromosome f and virtual node j in a VNR respectively. We consider all virtual node mappings are similar. Therefore, their order in a chromosome can be assigned in an arbitrary manner. Once assigned, all the chromosomes will follow the same order. To enhance the capability and reduce execution time, we initially generate a feasible node pool where all eligible substrate nodes meeting the least virtual node requirements of the VNR are collected. Each virtual node in a VNR is consecutively mapped by a substrate node which is randomly selected in the initial node pool, meeting resource requirements of the virtual node (Eq. (3.1)-(3.4)). We assume that mapping virtual nodes is conducted in random to prevent the possible premature convergence problem.

Since mappings of all virtual nodes are determined, various path searching methods as described in 4.2.2 are applied to explore feasible mapping solutions of associated virtual link requests, based on the already-found node mapping information. These physical paths must meet all virtual resource requirements to become potential link mapping solutions. When a feasible solution for the whole VNR has been successfully defined, a chromosome will be established. This is an ideal scenario. In reality, there might have a situation that a virtual link request cannot find any feasible path due to network congestion, and such pair of virtual nodes obviously need to be remapped. In worse cases, more than one virtual link request might fail to be embedded, so it is

desired to have an efficient mechanism to intelligently remap these requests concerning the minimal number of virtual nodes associated with their corresponding substrate paths that should be revisited.

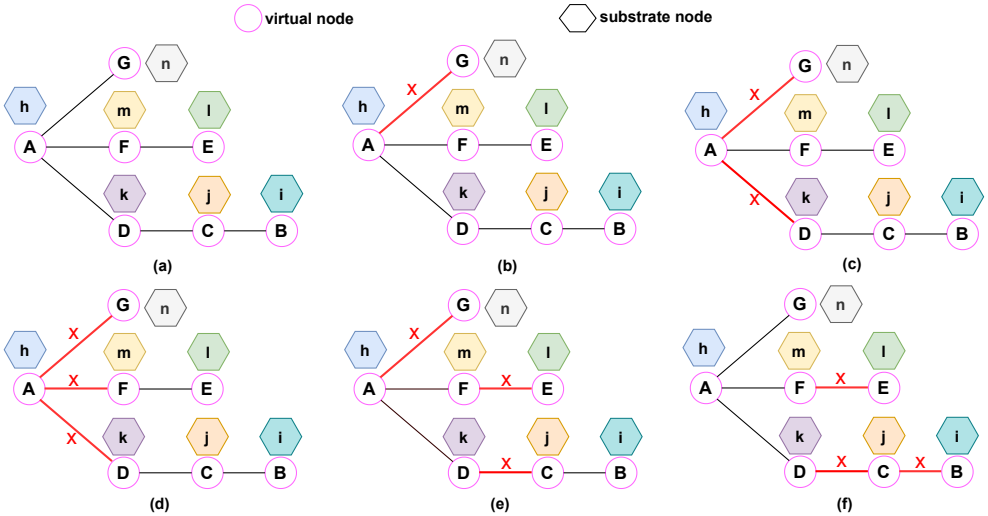


Figure 4.2: *Examples of infeasible virtual link mappings*

Let's take a real VNR with 7 virtual nodes and 6 virtual links for an example as demonstrated in Fig. 4.2 with several practical scenarios. Fig 4.2a is an accepted solution since all virtual nodes and links have been successfully found their feasible mappings. Next figure illustrates a simplest failure with a single failed VLiM from $A-G$. In this case, the substrate nodes h and n, node mappings of virtual nodes A and G, respectively, need to be remapped. If n is selected, we would remap only one virtual link $A-G$; otherwise, we have to remap three virtual links $\{A-G, A-F, A-D\}$. Fig. 4.2c complicates the same problem a bit with two failed virtual links $\{A-G$ and $A-D\}$. We possibly have three options for remapping including substrate nodes {h, k and n}. Each selection would give the same number of remapped virtual links but h should be chosen instead of k and n since least virtual nodes need to be reconsidered. Similarly, h should be selected for remapping in Fig. 4.2d, even there are three failed link mappings including {A - G, A - F and A - D}. If these all already-mapped virtual nodes or virtual nodes {D, F and G} are revisited, we need to remap at least five virtual links. Otherwise, when the substrate node h, a node mapping solution of virtual node A, is reconsidered, the remapped virtual links could be only three. In Fig. 4.2e, substrate nodes k/j, l and n should be remapped whereas physical nodes j

Algorithm 3 Distributed and parallel GA-based algorithm

Input:
    + Online VNR
Output:
    + Efficient mapping solution for the VNR
function `Initial path generation`
    k-shortest path algorithm is deployed to explore possible paths between each pair of source and destination nodes focused on hop-count factor.
end function
function `Initial node pool generation`
for each $n^s \in N^s$ **do**
    if $R_N(n^s) \geq c(n_i^{v_m})$ and not assigned before [$\triangleright$ $n_i^{v_m}$ ***is the virtual node with minimal CPU requirement.***] **then**
        add n^s to initial node pool.
end function
procedure VIRTUAL NETWORK EMBEDDING
        Step 1: *implement* ***Genetic Algorithm*** *in distributed and parallel machines as detailed in* ***Algorithm 4***
        Step 2: *Synchronize achieved incumbents among parallel machines.*
        Step 3: *Select the global incumbent solution based on sum of fitness values (E.q. 3.33)*
    return *mapping solution for a VNR*
    Update substrate resource information

Algorithm 4 Genetic Algorithm with Conciliation Strategy at each parallel machine

Input:
+ *Online VNR*
+ *Initial path pool*
Output:
+ *The mapping solution for the VNR*
procedure GENETIC ALGORITHM OPERATIONS
Initial population generation
while ($i \leq M$ chromosomes) **do**
Step 1: *Generate a chromosome that has $\mathcal{N}$ genes. Each gene is the mapping solution of a virtual node which is randomly selected in Initial Node Pool Generation.* [▷ ***substrate nodes must meet resource requirements to become***] [**feasible solutions as explained in Section 3.2.1**]
Step 2: *When the chromosome formed, the link mapping is deployed to find the mapping for each virtual link*
Step 3: *Heuristic Conciliation algorithm (Algorithm 5) is apply for a possible set of failed virtual links after Step 2*
Step 4: *Remap virtual nodes and failed virtual links accordingly*
while ($j \leq maxIterations$) **do**
Selection *Select two random chromosomes as parents*
Crossover *Swapping parental genes beginning from the random crossover point to the last one*
Mutation *A new child is generated by replacing an existing gene of the parental chromosome with a new one in random*
Output the best solution known as incumbent based on fitness values (E.q. 3.34)

and i can be re-embedded in Fig. 4.2f due to least remapping. Inspired by this idea, we present a heuristic conciliation algorithm to handle this problem as detailed in Algorithm 5 where lines [10-22] attempt to quantify potential nodes for re-mapping. For better understanding, let take an example in Fig. 4.2f for our remapping strategy applied on the SN demonstrated on the top of Fig. 4.3. The VN (7 virtual node and 6 virtual links) shows in right corner of Fig. 4.3 while the SN can be abstracted on the left side. We have three failed virtual links ($F-E, D-C$) and $C-B$. After applying conciliation strategy, it is merely required to remap virtual nodes C and E instead of all virtual nodes in which old node mappings ($C \rightarrow j, E \rightarrow l$) are now remapped to new mappings ($C \rightarrow s, E \rightarrow p$). As a result, all virtual links are mapped successfully.

Algorithm 5 Heuristic Conciliation Algorithm

1: **Input:**
2: *Initial solutions of virtual node mapping*
3: *A set of failed virtual links after link mapping phase*
4: **Output:**
5: *Virtual nodes should be remapped*
6: **procedure** HEURISTIC CONCILIATION STRATEGY
7: **Step 1:** *Construct a map M_p of virtual nodes based on their presence in a set of failed virtual links*
8: **Step 2:** *Create a map M_n of virtual nodes based on their nearest neighbors* [▷ ***These steps can be done in advance prior this algorithm***]
9: *Initialize an array for remapped nodes*
10: **for** each infeasible virtual link **do**
11: **if** s_i^v or d_i^v NOT in the array **then**
12: **if** $M_p[s_i^v] > M_p[d_i^v]$ **then**
13: *add s_i^v into array*
14: **else if** $M_p[s_i^v] < M_p[d_i^v]$ **then**
15: *add d_i^v into array*
16: **else**
17: **if** $M_n[s_i^v] > M_n[d_i^v]$ **then**
18: *add d_i^v into array*
19: **else**
20: *add s_i^v into array*
21: **return** *node array*

4.2.4 Working Nodes

As illustrated in Fig 4.1, each paralleled machine is independently running a GA algorithm which is consisted of four major operations: population initialization, selection, crossover and mutation.

Population Initialization: Each working machine starts with a population initialization step. Denote $\mathcal{M}$ as a set of chromosomes. Each chromosome includes $\mathcal{N} = |N_i^v|$ genes that represent potential mapping solutions of all virtual node requests in a VNR. An initial population $\mathcal{P}_n$ ($\mathcal{M}$x$\mathcal{N}$ size) at a working machine is generated as described in Section 4.2.3.

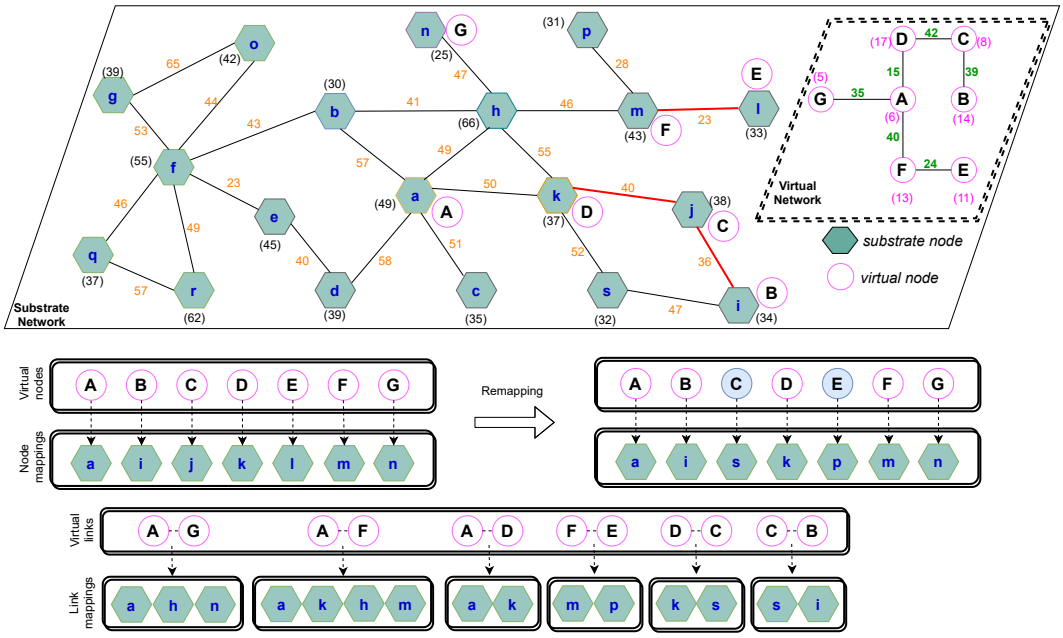


Figure 4.3: *Examples of Conciliation and remapping*

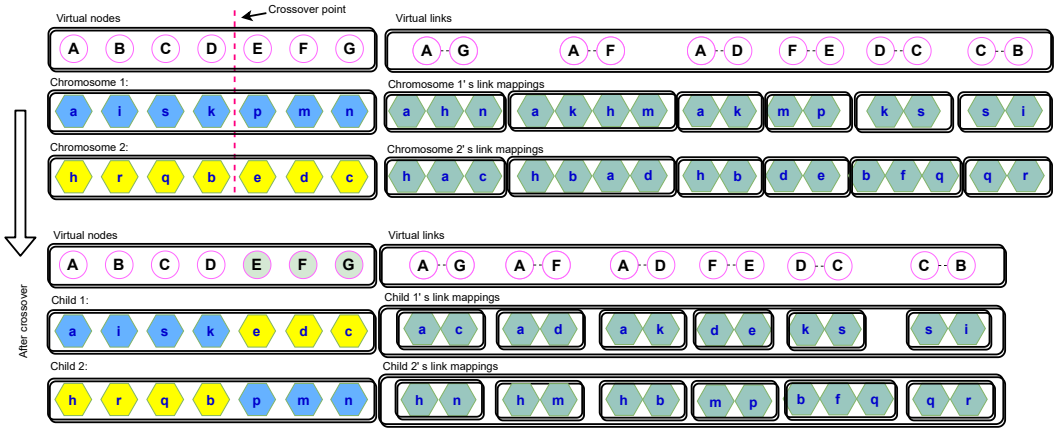


Figure 4.4: *Crossover operation*

$$\mathcal{P}_n = \begin{bmatrix} \mathcal{C}_1^n \\ \mathcal{C}_2^n \\ \vdots \\ \mathcal{C}_f^n \\ \vdots \\ \mathcal{C}_{\mathcal{M}}^n \end{bmatrix} = \begin{bmatrix} \Phi_1^1 & \cdots & \Phi_1^j & \cdots & \Phi_1^{\mathcal{N}} \\ \Phi_2^1 & \cdots & \Phi_2^j & \cdots & \Phi_2^{\mathcal{N}} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \Phi_f^1 & \cdots & \Phi_f^j & \cdots & \Phi_f^{\mathcal{N}} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \Phi_{\mathcal{M}}^1 & \cdots & \Phi_{\mathcal{M}}^j & \cdots & \Phi_{\mathcal{M}}^{\mathcal{N}} \end{bmatrix} \tag{4.1}$$

$$\mathcal{P}_e = \begin{bmatrix} \mathcal{C}_1^e \\ \mathcal{C}_2^e \\ \vdots \\ \mathcal{C}_f^e \\ \vdots \\ \mathcal{C}_{\mathcal{M}}^e \end{bmatrix} = \begin{bmatrix} \Psi_1^1 & \cdots & \Psi_1^j & \cdots & \Psi_1^{|L_i^n|} \\ \Psi_2^1 & \cdots & \Psi_2^j & \cdots & \Psi_2^{|L_i^n|} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \Psi_f^1 & \cdots & \Psi_f^j & \cdots & \Psi_f^{|L_i^n|} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \Psi_{\mathcal{M}}^1 & \cdots & \Psi_{\mathcal{M}}^j & \cdots & \Psi_{\mathcal{M}}^{|L_i^n|} \end{bmatrix} \tag{4.2}$$

As such, each chromosome in population is initially formed and then the path ranking mechanism is deployed for VLiM based on node mapping solutions that have been already achieved. A conciliation mechanism is used to handle possible failures of virtual link mappings with a goal of minimal virtual node/link mappings revisited. Since feasible solutions of all virtual nodes and links are successfully obtained, the

chromosome for a VNR is properly established. It appears that if any node mapping changes, the associated link mapping is affected accordingly.

New generations In this work, we randomly select chromosomes to be parents for generating their children. Selected parents produce new generations as a result of crossover and mutation operations, which consequently makes the mapping solutions evolved after the number of iterations. After those operations, the major problem is seeking feasible physical paths regarding to changes of new nodes. Similar to Section 4.2.3, the path ranking method is implemented to explore link mapping solutions for new generations towards new pairs of substrate source-destination nodes. If no feasible paths found on new generations, they will be discarded. Otherwise, the newly generated generations will be updated into the population to enhance the population diversity and therefore increase the possibility of approaching efficient mapping solutions.

Crossover: in this operation, parental chromosomes are combined in order to create new offspring for next generations. $\mathcal{C}_s$ and $\mathcal{C}_r$ represent the selected chromosomes where their indexes within the initial population are s and r respectively. j^c is the crossover point which is randomly chosen between any positions within $\mathcal{N}$ length. We denote $\mathcal{C}_{(\mathcal{M}+1)}$ and $\mathcal{C}_{(\mathcal{M}+2)}$ as new generated chromosomes In crossover, offspring is typically produced by exchanging genes between the selected parents beginning from the random crossover point j^{c+1} to the end of chromosomes as demonstrated in (3.30). At this stage, node mappings have been changed, link mappings are revisited by the path searching methods accordingly. If new generations are feasible, they will be updated into the existing population.

Suppose we select two chromosomes from the population as parents where one of them was used to demonstrate the conciliation strategy in Section 4.2.3. As depicted in Fig. 4.4, the random crossover point equals to 4, the node mappings of virtual nodes E, F and G will be swapped between two parents in order to generate next generations under two children. After new node mappings determined, we need to examine their corresponding link mappings. We can recognize that new link mappings

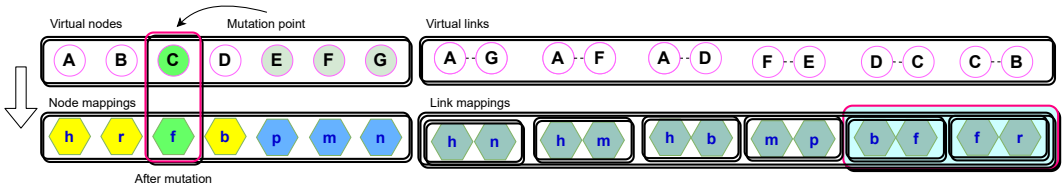


Figure 4.5: *Mutation operation*

are much better than the previous ones, which means that our children are evolved.

$$\mathcal{P}_n = \begin{bmatrix} \mathcal{C}_1^n \\ \vdots \\ \mathcal{C}_s^n \\ \vdots \\ \mathcal{C}_r^n \\ \vdots \\ \mathcal{C}_{\mathcal{M}}^n \\ \mathcal{C}_{\mathcal{M}+1}^n \\ \mathcal{C}_{\mathcal{M}+2}^n \end{bmatrix} = \begin{bmatrix} \Phi_1^1 & \cdots & \Phi_1^{j^c} & \Phi_1^{j^c+1} & \cdots & \Phi_1^{\mathcal{N}} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \\ \Phi_s^1 & \cdots & \Phi_s^{j^c} & \Phi_s^{j^c+1} & \cdots & \Phi_s^{\mathcal{N}} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \\ \Phi_r^1 & \cdots & \Phi_r^{j^c} & \Phi_r^{j^c+1} & \cdots & \Phi_r^{\mathcal{N}} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \\ \Phi_{\mathcal{M}}^1 & \cdots & \Phi_{\mathcal{M}}^{j^c} & \Phi_{\mathcal{M}}^{j^c+1} & \cdots & \Phi_{\mathcal{M}}^{\mathcal{N}} \\ \Phi_s^1 & \cdots & \Phi_s^{j^c} & \Phi_r^{j^c+1} & \cdots & \Phi_r^{\mathcal{N}} \\ \Phi_r^1 & \cdots & \Phi_r^{j^c} & \Phi_s^{j^c+1} & \cdots & \Phi_s^{\mathcal{N}} \end{bmatrix} \tag{4.3}$$

Mutation: This operation typically adopts a modification on a parent to generate a new offspring. Mutation samples the broad solution space and improves the searching efficiency, preventing solutions from falling into the local optima. A random gene of the selected parent is replaced by a new gene to create a new child. The gene used for replacement must explicitly meet the resource constraints. Similar to crossover, path searching mechanisms are implemented to find new link mappings due to node mappings changed. If feasible, new generation is updated into population. Let denote j^m as a random mutation point and $\Phi_{r'}^{j^m}$ is new gene that replaces the existing one in $\mathcal{C}_{(\mathcal{M}+1)}$. In this operation, new embedding solution $\mathcal{C}'_{(\mathcal{M}+1)}$ after replacement is presented as $\mathcal{C}'_{(\mathcal{M}+1)} = [\Phi_s^1 \cdots \Phi_{r'}^{j^m} \cdots \Phi_s^{\mathcal{N}}]$. As illustrated in Crossover, we achieved new children which are better than their parents. Suppose the second child is randomly chosen for the mutation, where the mutation point is selected in random, equal to 3. Old node mapping $(C \rightarrow q)$ is randomly replaced by $(C \rightarrow f)$, which results in

new link mappings for virtual links ({D-C} and {C-B}). Specifically, substrate paths $(b \rightarrow f \rightarrow q)$ and $(q \rightarrow r)$ are now substituted by $(b \rightarrow f)$ and $(f \rightarrow r)$ respectively. Both new children are feasible and better than their parents, so they will be updated into the population.

4.2.5 Sorting and Terminations

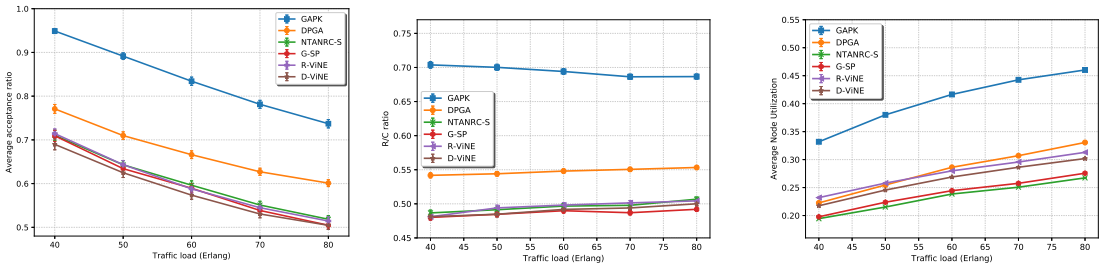
Sorting process selects the best embedding solution among the feasible ones based on their fitness values, and then it is conveyed to synchronization step for a global ranking. To reduce execution time, the master node terminates GA algorithms in running worker nodes if no better mapping solution is achieved within t times, where t denotes a termination parameter.

4.2.6 Synchronization and VNR Allocation

In this step, the best VNE solution of the corresponding VNR is determined by globally ranking the VNE solutions received from worker nodes, based on highest achieved FF values. As a result, if accepted the VNR is then allocated onto SN following the information of the virtual node and link mapping solutions obtained. The last step is updating residual network resources.

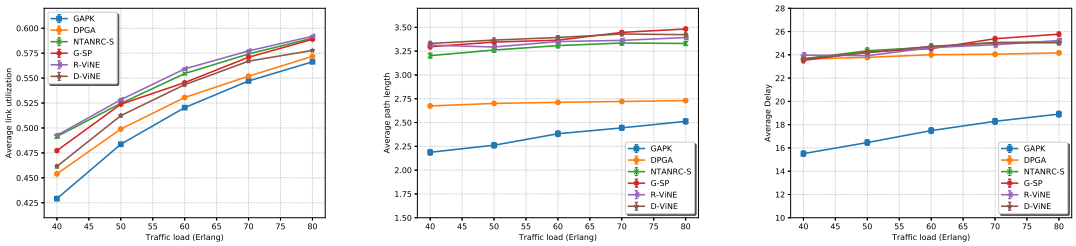
4.3 Evaluation and Numerical Results

We compare our proposed GA-RAN, GA-SEQ, GA-STP and GAPK algorithms with several state-of-the-art VNE competitors: IDPA [13], DPGA [129], NTANRC-S [69], D-ViNE, R-ViNE, and G-SP [12] and MCTS [116] on several evaluation metrics comprising average acceptance ratio, average revenue and cost, average R/C, average node and link utilization and finally average delay. These are the most popular and important evaluation criteria for evaluating the efficiency of any VNE algorithm. We select [12] since it is widely known as one of the most highly-cited papers in VNE. Moreover, NTANRC-S, performed best in [69], is the latest representative of the node ranking methods that exploits multiple topology attributes and the global network resources for VNoM in VNE. Recently, our work in [13] and [129] have confirmed that VLiM contributes a critical role in approaching an efficient solution for VNE



(a) **(b)** **(c)**

Figure 4.6: **(a)** Avg. acceptance ratio **(b)** Avg. R/C ratio **(c)** Avg. node utilization



(a) **(b)** **(c)**

Figure 4.7: **(a)** Average link utilization **(b)** Average path length **(c)** Average delay

problem. Finally, MCTS is the reinforcement learning-based algorithm that deploys Monte Carlo Tree Search to explore the action space [24].

4.3.1 Simulation Setup

We also deploy a discrete-event simulator to assess our proposed VNE solution. Most of simulation parameters such as network topology generator, substrate or virtual resource capacity, arrival distribution, life time are similar to [13] and [12]. However, the load of VN requests is quantified by $\frac{\lambda}{\mu}$ Erlangs. The error bars were very small due to a large number of samples used, which proved that our simulation results were obviously reliable. For better presentation, we plotted figures with different colors and markers.

In next sections, we discuss on the simulation results between our proposed VNE algorithms and state-of-the-art VNE algorithms. Then, a comprehensive analysis in performance between the joint node-link algorithms themselves is also presented. Due to different characteristics between the path ranking method and the other

link mapping algorithms and similar performance results between GAPK and GA-SEQ [132], we make two groups for evaluation purposes.

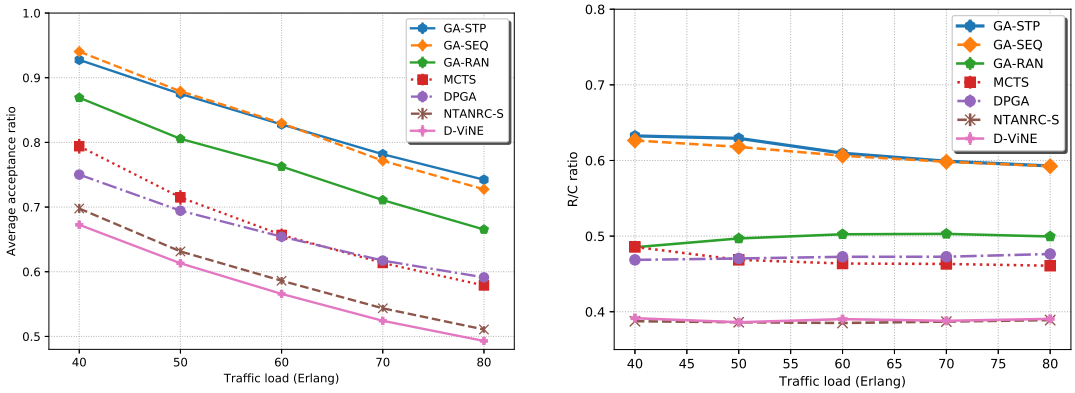
4.3.2 Joint Node-Link Mapping with Path Ranking Method

Simulation results are intensively illustrated in Figure 4.6 and 4.7. Our proposed GA-based algorithm, namely ***GAPK***, achieved highest acceptance ratio by accepting more VNRs with considerably lower costs than all competitors, leading to highest average revenue to cost ratio as illustrated in Fig 4.6b. In fact, higher acceptance and revenue to cost ratios is desirable for any VNE algorithm, so our proposed solution has proved its efficiency. Specifically, GAPK improved the acceptance ratios of DPGA for more than 17.81% and 13.59% at 40 and 80 Erlang respectively. GAPK significantly performed better than NTANRC-S and R-ViNE (the best algorithm in [12]) for more than 22% at the highest traffic load as shown in Fig. 4.6a. Our proposed joint node-link VNE solution gained 24.8%, 34.4% and 36.1% better average revenue to cost ratios than those of aforementioned algorithms at the same traffic load as depicted in Fig 4.6b. By accordingly accepting more VNRs, node utilization of GAPK is remarkably higher 10% up to 20% than its all rivals at all traffic load (Fig. 4.6c).

Moreover, it is observed that GAPK and DPGA extremely produced short average path length in a comparison with other algorithms as illustrated in Fig. 4.7b, which indeed contributes to low embedding cost and enhanced average link utilization (Fig. 4.7a). The reason behind these appealing outcomes is that we take hop-count factor into account in a multi-constrained FF for selecting the optimal VNE solution. Due to less bandwidth consumed to map virtual link requests, abundant residual bandwidth enables to accept incoming VNRs confirmed by Fig. 4.6a. In addition, we evaluate all compared algorithms in average delay metric. As depicted in Fig. 4.7c, GAPK empowered by an intelligent GA algorithm effectively explored physical paths with very low delay, reducing network resource fragmentation by preferring neighboring substrate nodes for mapping VNs.

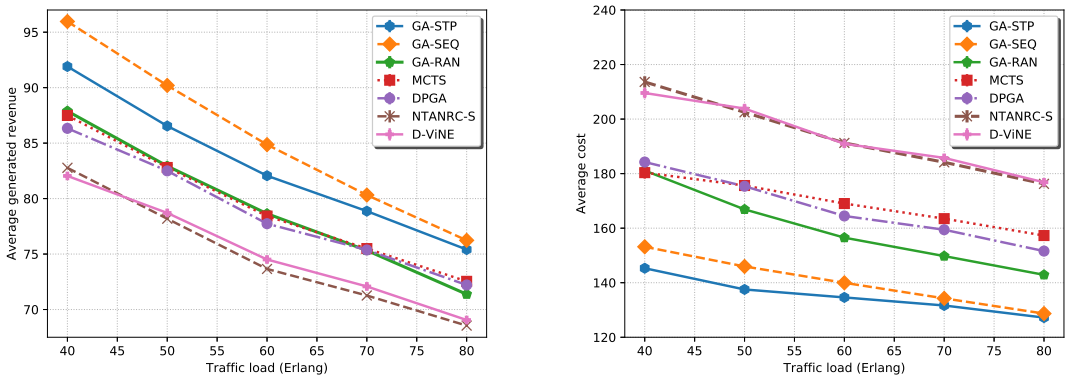
Furthermore, we conducted the joint node-link VNE mapping on sequential and parallel schemes and then compared execution time between them to quantify time reduction with parallel operation. Distributed and parallel paradigm as shown in Fig.4.1 took $1.2s$ to finish processing a VNR in average while the sequential counter part needed more than $10.1s$ to accomplish the same task. With regard to time complexity and convergence, interested readers may refer to Section 3.3 for further

theoretical analysis.



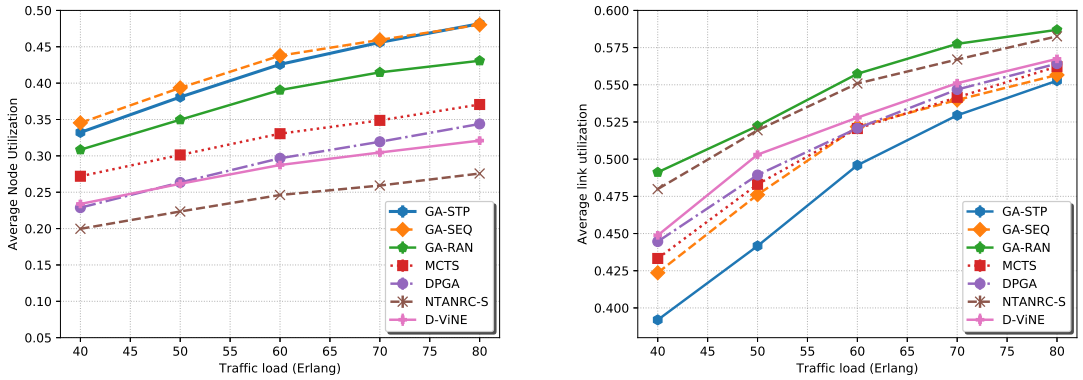
(a) **(b)**

Figure 4.8: **(a)** Average acceptance Ratio **(b)** Average revenue to cost ratio



(a) **(b)**

Figure 4.9: **(a)** Average generated revenue **(b)** Average embedding cost



(a) **(b)**

Figure 4.10: **(a)** Average node utilization **(b)** Average link utilization

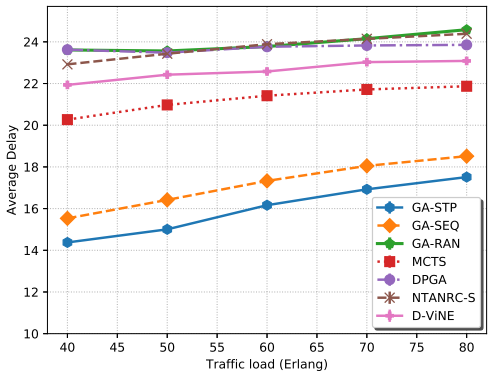
4.3.3 Joint Node-Link Mapping with Several Path Searching Algorithms

Joint node-link algorithms vs State-of-the-art VNE algorithms

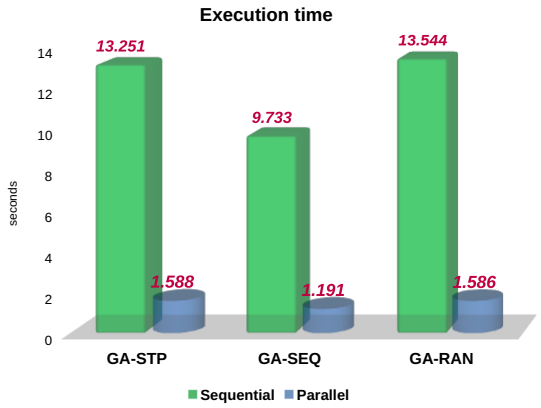
Fig. 4.8a illustrates the average VNR acceptance ratio which is one of the most important metrics to evaluate the embedding capability of different VNE algorithms. It is observed that the average VNR acceptance ratio of all algorithm decays with the increasing of network loads due to infinite substrate resources.

Our joint node-link VNE algorithms, including GA-STP [133], GA-SEQ [132] and GA-RAN [133] outperformed all competitors in the average acceptance ratio performance. Specifically, GA-STP, GA-SEQ and GA-RAN achieved better average acceptance ratios than MCTS (the best algorithm amongst compared algorithms in general) for more than $\{13.35\%, 14.63\%, 7.5\%\}$ and $\{16.35\%, 14.87\%, 8.65\%\}$ at 40 and 80 Erlangs respectively. They significantly performed better than DPGA, NTANRC-S and D-ViNE for more than 7.4% up to 24.93% at the highest traffic load.

Furthermore, the proposed GA-based algorithms produced higher average revenue and lower embedding cost, which leads to higher R/C as illustrated in Fig. 4.9a, 4.9b and 4.8b respectively. In details, our best joint node-link VNE solution (GA-STP) gained $30.23\%, 34.98\%, 63.23\%$ and 61.56% better average R/Cs than those of aforementioned algorithms at the lowest traffic load. Its overwhelming dominance towards this performance metric was retaining over various arrival rates as depicted in Fig 4.8b. These results have again confirmed by better average revenue and embedding



(a)



(b)

Figure 4.11: **(a)** Average delay between the proposed algorithms **(b)** A comparison of execution time

cost of our approach in Fig. 4.9a and 4.9b. The reason behind these great results is that the proposed VNE algorithms reduced resource fragmentation problem caused by possibly inefficient embedding. It means that adjacent virtual nodes of a given VNR were mapped onto substrate nodes that were geographically close from each other in the SN, decreasing inappropriate substrate resource consumption. In fact, higher average acceptance and R/Cs are the most desired target for any VN mapping algorithm, and our proposed approaches have successfully proven its great efficiency. With the increasing network loads, node and link utilization was expected to increase as depicted in Fig. 4.10a and 4.10b. Specifically, the node utilization of our joint node-link algorithms was remarkably higher at least 13.41% up to 77.83% than their all rivals over various traffic loads (Fig. 4.10a). Our algorithms were indeed capable of accepting more VNRs compared to other heuristics, considering multiple crucial embedding factors during mapping. When the network loads were increasing, joint node-link heuristics still successfully embedded given VNRs by exploiting physical resource capacity more efficiently. However,the link utilization did not exhibit the same behavior for all algorithms because of different link mappings. Due to the random searching strategy, GA-RAN might not able to guarantee the link mapping solutions with short path lengths, so it can be understandable that GA-RAN did not have an advantage on the link utilization as shown in Fig. 4.10b. On the other hand, GA-STP and GA-SEQ exceedingly leaded this performance metric over the other algorithms thanks to effective path searching approaches as described in Section

4.2.2. Additionally, it was expected that GA-STP had the best performance in average delay since we assumed that the shortest path method used in GA-STP is aimed at minimizing the average delay in previous sections, following by GA-SEQ which respects to the hop-count factor. Both empowered by an intelligent GA algorithm effectively explored substrate paths with low average delay, reducing network resource fragmentation by preferring neighboring substrate nodes for mapping VNs. Particularly, GA-STP and GA-SEQ were better than MCTS, the best delay performance among the compared algorithms, for at least 15.33% up to 29.07% as depicted in Fig. 4.11a.

These dramatic results of our GA-based approaches derive from the fact that the proposed algorithms consider the coordination between node and link mapping stages at the same time. The novel conciliation mechanism not only handles the remapping strategy intelligently by minimizing the number of virtual nodes and virtual links for remapping, but also reduces the possibility of missing out optimal solutions due to remapping all virtual nodes. Moreover, GA algorithm efficiently explored the search space to approach optimal VNE solutions driven by an efficient multi-objective fitness function. Due to less bandwidth consumed to embed virtual link requests, abundant residual bandwidth enables to accept more incoming VNRs verified by Fig. 4.8a. The appealing results of our joint node-link GA-based algorithms are indeed a real challenge for any VNE algorithm we suppose.

Amongst Joint Node-Link Algorithms

In this paper, we proposed three joint node-link algorithms according to different path searching methods. They were integrated into GA algorithm for embedding VNRs using the same fitness function to drive the algorithm to optimal VNE solutions. GA-STP with shortest path searching performed best among them following GA-SEQ and finally GA-RAN. Although, GA-STP did not clearly show its complete dominance over GA-SEQ in the acceptance ratio, average revenue, and node utilization; GA-STP incontrovertibly outperformed all in the rest of evaluation criteria. With low network loads, GA-SEQ was slightly better than GA-STP for accepting more VNRs but when the number of VNRs were increasing, GA-STP became more effective in embedding. This can explain why GA-SEQ was little better than GA-STP in node utilization in Fig. 4.10a. In contrast, GA-STP demonstrated its exceptional performance on virtual link embedding through proportional embedding cost, average link utilization and average delay by leveraging the shortest path method. Specifically, GA-STP

was better than its counterpart, GA-SEQ, in the same aforementioned performance metrics for 5.42%, 3.16% and 9.41% respectively at the arrival rate of 40. Both tended to be asymptotic when the network became more and more congested. In fact, GA-STP handled network resource fragmentation better than GA-SEQ since it preferred substrate paths that owned lower propagation delays for embedding VNRs, which properly improved substrate resource consumption. On the other hand, GA-RAN performed worse among three since it relied on the arbitrary searching mechanism in the path database instead of the sequential search like GA-SEQ where the feasible solution found was the one had the smallest hop-count feature. Hence, GA-RAN might approach the link mapping solutions with longer path lengths as illustrated in Fig. 4.10b. Furthermore, we conducted the joint node-link VNE mapping algorithms on different sequential and parallel schemes and then compared execution time between them to quantify the time reduction towards parallel operation. The distributed and parallel paradigm as shown in Fig.4.1 was deployed for the parallel operation. Average execution time of the proposed algorithms for processing a VNR is shown in Fig. 4.11b. As such, GA-SEQ was fastest compared to the other algorithms in both experiment schemes. Towards sequential operation, GA-SEQ was faster than GA-STP and GA-RAN for more than 26% and 28% respectively. In this scheme, GA-RAN that is based on random mechanism needed more time to seek for feasible solutions while GA-SEQ due to its simplicity took least time to finish processing a VNR. On the other hand, parallel scheme reveals that shortest path method and random mechanism most likely achieved the same execution time, which means that GA-RAN exploited the distributed machines approaching VNE solutions efficiently with the random manner. GA-SEQ is still a winner since it was faster than the other counterparts for more than 25%. Although, GA-STP performed best within the set of proposed joint node-link VNE algorithms in terms of performance, if we need a more rapid mechanism, an alternative choice could be GA-SEQ.

4.4 Chapter Summary

In this chapter, we propose joint node and link embedding approaches relied on GA algorithm for simultaneously solving virtual node and link mappings in multiple stages in the population initialization of GA algorithm, where the link mapping is relied on various path searching methods. When the node mappings are changed, the link

mappings are accordingly altered. A heuristic conciliation mechanism is then used to tackle infeasible link mappings due to inappropriate virtual node embedding in GA operations. Furthermore, we deploy our proposed algorithms on a distributed and parallel operation scheme in order to reduce time execution and then we make a time comparison of the proposed VNE solutions running on sequential and parallel operations. The extensive evaluation indicates that our joint node-link combination in a single VNE mapping based on GA-based approaches outperforms most of state-of-the-art heuristic VNE algorithms in entirely indispensable evaluation metrics we adopted.

Chapter 5

Collaborative Parked Vehicle Edge Computing Framework for Online Task Offloading

As cited in 1.2, we would like to leverage the proposed GA-based approach to solve the resource allocation problem in another different research field in order to examine the diverse capability of our proposal.

5.1 Introduction and Motivation

Due to a rapid development of the Internet of Everything (IoE), the number of smart devices connected to the Internet has been immensely increasing. Following the statistics from Internet Data Center (IDC), the number of terminals and devices connected to the network would exceed 50 billion by 2020 [134], generating massive growth of data. An explosive growth of data traffic with heterogeneous computation demands, either sensitive or tolerable to higher latency, will lead to a heavy burden of network workloads during peak hours. Unfortunately, typical cloud computing is no longer capable to support the diverse needs of today and future networks for data processing. Edge computing has urgently emerged as the solution candidate for the next generation networks, and has widely become a major research topic. The primary idea of edge computing is to extend the cloud computing capabilities by bringing computing services close to end users at the edge.

In fact, the coexisting of remote cloud and edge computing is the most dominant form for task execution, and the offloading tasks are not always delay-sensitive in

reality. Thus, incoming tasks that do not match well with the cloud in terms of delay requirements can be offloaded to the network edge. Due to limited computing capacity of edge computing, when more applications demand sensitive-latency tolerance during peak hours, the network becomes congested.

On the other hand, last decade has witnessed an increasing number of vehicles that are anticipated to reach 2 billion by 2035 [135]. Many of which are equipped with modern on-board computers and sensors to offer advanced features such as autopilot, self-parking, radar, automatic safety systems. Notably, studies have shown that 70% of individual vehicles almost spend 95% of time for parking in parking lots, home garages or street parking spaces [135] [136]. Those on-board computers with level-four autonomous driving support can cost tens of thousands of dollars but the expected utilization of vehicles is extremely low. For example, the average driving time per day in America was only 50.6 minutes according to the data survey of AAA Foundation for Traffic Safety in 2016 [137]. These statistics show that on-board computers are idle in most of the time, suggesting that significant computation resources can be exploited for provisioning additional services. The overlooked resources of PVs can become ideal candidates for transient resources leveraged by Mobile Edge Computing (MEC). By edge computing, the conventional computation and storage services offered by the remote cloud are now extended to the edge of the network. With the introduction of PVs, the strength of edge is now doubled, but they complicates the offloading problem through determining the proper network resources to execute a given task. Moreover, incoming tasks can be classified into delay-sensitive (e.g. augmented reality, video surveillance, mobile gaming, transportation, live streaming, robot collaboration, autopilot) and delay-insensitive with rigid computational requirements [138]. Therefore, we need efficient algorithms to embed online heterogeneous tasks onto the collaborative framework between cloud-edge and PVs effectively. Moreover, application vendors have little chance to negotiate the embedding service costs since these services are often provided by Service Providers (SPs) with a fixed amount of service charges. Additionally, owners of those smart vehicles cannot get any benefit from their powerful on-board computing resources while parking.

Little work has been done to apply a generic container orchestration system to edge computing enhanced by PVs. Container orchestration allows PVs to efficiently run multiple task replicas simultaneously with fast boot-up and short termination time. Such agility is important due to PVs' limited and arbitrary parking duration,

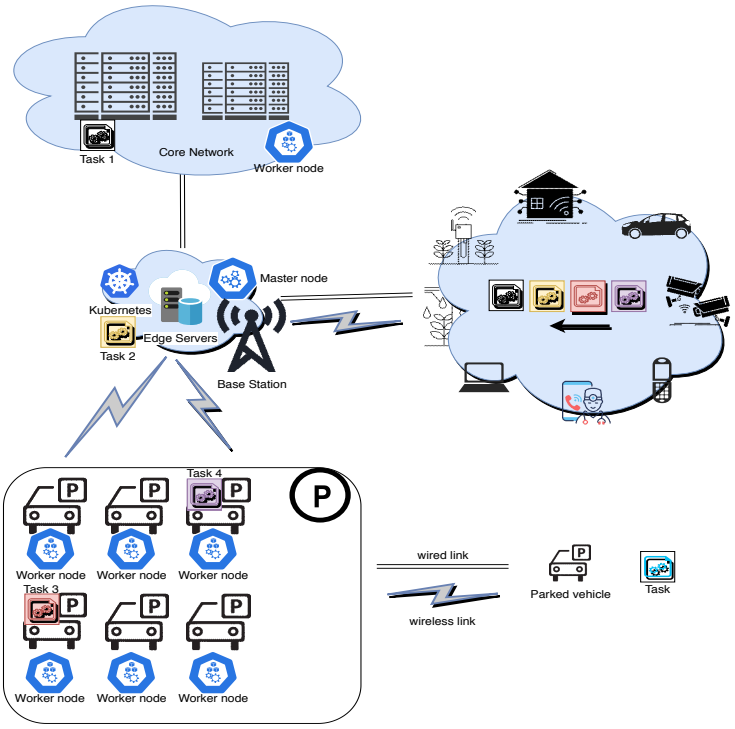


Figure 5.1: Edge computing architecture integrated with parked vehicles enabled by Kubernetes.

enabling them to reliably execute tasks. Containers may also require lower hardware requirements, operation costs and support resource isolation since each container processes given tasks independently. The de facto industrial standard container orchestration framework (e.g. Kubernetes [139]) offers advanced features such as auto-scaling, self-healing and security policies. Kubernetes will run the task workload by allocating containers into pods to run on worker nodes. When a PV node fails, Kubernetes automatically reschedules the requested tasks onto another healthy node. The control plane in the master node manages the worker nodes and the pods in the corresponding cluster, and determine global decisions about the cluster (e.g. scheduling, scaling). Kubernetes platform will tackle the uncertainty of PVs' mobility and network interruptions due to advanced automation techniques. When online tasks dynamically arrive at the master node with stringent constraints (e.g. CPU, bandwidth, delay tolerance), kube-scheduler in the control plane of the master node creates pods and assigns the node for them to run on. More details about Kubernetes can be found in [139].

In this section, we present EdgePV, a novel collaborative architecture that enables a flexible task execution during peak hours, where PVs become computing nodes to expand the existing resource capacity at the edge. Owners of PVs may sell their resources cheaper than ISPs for utility or rewards. These can be converted to accumulating rewarded points for shopping, gas, parking tickets and so on. Our proposed architecture investigates the possibility of an integration between the legacy cloud computing, an emerged edge paradigm and PVs. Moreover, we advocate the possibility of integrating Kubernetes-based platform into this model that is aimed at improving QoS and security concerns. As illustrated in Fig. 5.1, master node is placed at edge node, managing the worker nodes which include the computation resources from the cloud, edge server itself and mobile PVs to handle online task offloading. Those are adopted Kubernetes system forming a container orchestration cluster to provide several advanced features such as load-balancing, auto-scaling, auto-healing, resource management, security, etc,. The proposed collaborative framework aims to enhance the elasticity and agility of the existing network architecture to diminish any possible service disruption due to the mobility of PVs. It is also envisaged that PVs could be completely electric-based soon, and they would be automatically charging while parking. Moreover, D2D technology can be integrated into this collaborative infrastructure. Still, it is beyond the scope of this research since its main goal is to tackle the online task offloading problem.

The online task offloading problem of the proposed collaborative framework is formulated as a BIP problem in Section 5.3 aimed at minimizing overall offloading costs while maximizing accumulative utility, that is computationally intractable to solve exactly and widely known to be NP-hard. We simulate the model with the random mobility behaviors of PVs with random and dynamic task arrivals, and compare a set of scheduling algorithms in performance evaluation. Thus, we will propose a heuristic algorithm to obtain sub-optimal results by applying min-max algorithm (denoted as M&M) in aim of minimizing offloading costs while maximizing utility or rewards when given tasks. For simplification, the simple algorithm would not consider where the replicas of a task are offloaded. This strategy would possibly be a weakness of such proposal. For example, if all task replicas are allocated to a single node (e.g. PVs), but this node gets an unexpected failure (e.g. outage, sudden vehicular leaving). Services provided through running such containerized task will be interrupted.

Furthermore, inspired from the success of the distributed and parallel computing

framework relied on GA algorithm in Chapter 3 and 4, we leverage the proposed scheme introduced in Section 3.3 to address the task offloading problems, meeting rigid task requirements, lower costs and high reliability. This model takes the reliability of offloading solutions into account by intelligently considering where and how to allocate replicas of a task to the collaborative computing framework. This flexibly allows SPs to determine a least proportion of replicas possibly running on different worker nodes (e.g. 50%). In both proposed algorithms, owners of PVs who are selling their resources can obtain accumulating reward points (user utility) that can be converted to parking tickets, gift-card, shopping vouchers, gas, and so on.

We intend to compare proposed algorithms with several baseline algorithms (e.g. Kubernetes default scheduler with filtering and scoring procedures, task offloading with random policies, branch and bound policy). Three major performance metrics are also evaluated including task average acceptance ratios (A/R), average costs and average accumulated utility/rewards. Additionally, we investigate the PV availability on the performance of accepting online tasks reflected by the acceptance ratios on various arrival rates. These information is obviously important for the network planners to target Key Performance Indicators (KPIs) by adopting proper offloading strategies.

5.2 Related Work

PVs as infrastructure that extend the computing environment for computation, communication and storage have received significant attentions in recent years. Enabling PVs to vehicular cloud computing in Internet of Vehicle system are studied in [140–145].

Arif et al. in [140] considered the simplest deployment of a vehicle cloud (VC) assisted by PVs in a long-term parking lot of an international airport. This research work predicted the parking occupancy to schedule network resources and then assigned the computation tasks. In addition, [141] introduced a multilayered vehicular data cloud architecture relied on cloud computing and IoT technologies. Two vehicular data cloud services including an intelligent parking and a vehicular data mining cloud services in IoT environment were also proposed. Similarly, the realizations of a VC built on PVs in a parking lot as a spatial-temporal network infrastructure for communication, computing and storage were explored in [142–144]. Additionally, another investigation that studied the feasibility of PVs as computation infrastructure was conducted in [145], proposing an incentive mechanism to offer rewards for PVs on

account of the parking duration as well as their energy consumption.

In addition, Hou et al. [146] proposed Vehicular Fog Computing (VFC) utilizing connected PVs as the infrastructure in order to provide real-time services at the edge with low latency. Authors also discussed four scenarios of exploiting PVs and slow moving vehicles as computation and communication paradigm. In similarity, fog computing implemented in Internet of Vehicle (IoV) systems to provision computation resources to end users with latency guarantee was presented in [147]. This proposed paradigm enabled to offload the real-time network traffic in fog-based IoV systems in aim of minimizing the average response time.

Recently, PVs proposed to become accessible edge computing nodes to conduct task execution, called by Parked Vehicle Edge Computing (PVEC) were introduced in [135, 137, 148, 149]. The authors in [135] cooperated Vehicle Edge Computing (VEC) servers and PVs to explore possible opportunistic resources in order to deal with computational tasks. Their study investigated the problem of user payment minimization simplifying budget or latency constraints, which would make the proposed scheme fall into sub-optimal. Similarly, adapting the current system state to calibrate the price constantly, [137] presented a dynamic pricing strategy to minimize the average cost while meeting the user QoS constraints. The authors in [148] proposed a container-based task scheduling scheme for PVEC in term of social welfare aspect concerning both users and PVs. To date, [149] recommended a delay-sensitive parked vehicular computing system that utilized softwarized block-chain technology for security and privacy. They only took PVs into account as an edge computing paradigm to handle task offloading and excluded the core cloud and edge server computing themselves.

5.3 Problem Formulation

We formulate the task offloading model concerning several network resource constraints at the edge of the network. The orchestration scheduler is placed in the edge server aside from the 5G base station (BS).

5.3.1 Offloading Model

In this research, we consider CPU, memory, and bandwidth resources. Three types of worker nodes, including cloud, edge, and PVs, exist in the network edge. These nodes can be generally connected to their master node placed at an edge server via

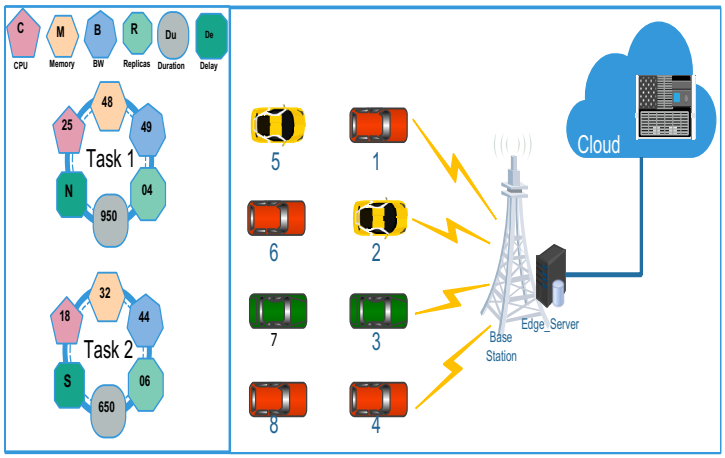


Figure 5.2: An example of task scheduling

different links. For instance, an optical link is deployed in the connection between the master node and the cloud node. In contrast, PVs join the edge network via wireless links where available vehicular bandwidth mainly relies on their corresponding distance to the 5G BS. Hence, a brief glimpse of network edge in a cluster can be conceived as a star topology where its root and leaves are the master node and its worker nodes, respectively. As depicted in Fig. 5.1, a generic outdoor parking lot is examined, where PVs are first necessitated to register their related information (e.g., owner's ID, available parking duration, license plate) and available vehicular resources (e.g., computing capacity, storage) with SPs. PVs might require sending or updating the current status information to the corresponding master node and their registered SPs when they arrive at the parking lot or just complete assigned tasks.

Our edge network can be modeled as a directed graph $G = (N, L)$ where N is a set of worker nodes, whereas L is the corresponding link. For example, Fig. 5.2 illustrates two requested tasks with several resource requirements, including CPU, memory, bandwidth, number of replicas, duration, and latency tolerance. Task 1 with the number of replicas can be scheduled at any nodes in the network (Cloud, Edge, or PVs) due to its latency-insensitive requirement. In contrast, task 2, a latency-sensitive type, even requires fewer resources, but its replicas could not be assigned in the remote cloud. For instance, if PVs are chosen, each potential PV is only allowed to handle a maximum of three replicas of task 2 to ensure service constancy. As mentioned, the edge server connects to the remote cloud via an optical link while PVs connect to the network edge via wireless links, denoted as l_c and l_v respectively. Each worker

node can create various pods to simultaneously host several containers of the task replicas with a QoS guarantee. A given task k in this model basically requires CPU $c(k)$, Memory $m(k)$, Bandwidth $b(k)$, tolerable maximum latency $t_m(k)$ and a number of replicas $\eth(k)$ demands. k_j denotes a j^{th} replica of the task $k \in K$, and $\sum k_j = \eth(k)$. Besides, each worker node $n_i \in N$ has its own resource capacity to handle a limited number of containers. CPU and memory capacity of i^{th} worker node are denoted as $C(n_i)$ and $M(n_i)$. K_c, K_e, K_p and K are sets of tasks successfully offloaded to the cloud, edge, PVs and network respectively, so $K = K_c + K_e + K_p$. Accordingly, the remaining CPU and memory capacity of a worker node can be calculated as below:

$$R_C^u(n_i) = C(n_i) - \sum_{k \in K} \sum_{j \in \eth(k)} c(k_j), \forall n_i \in N \tag{5.1}$$

$$R_M^u(n_i) = M(n_i) - \sum_{k \in K} \sum_{j \in \eth(k)} m(k_j), \forall n_i \in N \tag{5.2}$$

where u denotes the worker nodes (Cloud: c, Edge: e, PVs: $p \in P$), so $|N| = 2 + |P|$.

5.3.2 System Model

In our model, we assume the latency is inversely proportional to the residual resource capacity following the result of the M/M/1 queuing system.

Core network offloading latency

Network latency can be involved in the total time of data transmission and task execution. The formal is highly correlated to the residual bandwidth of the link l_c. Simultaneously, the latter relies upon "how busy" the cloud is to manage the offloaded tasks and other existing services. Thus, a large number of tasks offloaded to the cloud via l_c with less remaining network resources would technically increase the network latency, severely in peak hours. Due to the accelerating technologies in datacenters, the latency associated with writing and accessing data volumes from memory can be neglected. Cloud offloading latency $t_c(l_c)$ comprising the transmission latency and the processing latency can be calculated as below:

$$t_c(k) = \max_{j \leq |\eth(k)|} \{\frac{\chi_{k_j}}{\xi_c} + \frac{\chi_{k_j} f_{k_j}}{R_C^c(n_i)} + \frac{d}{v} + T_h\} \leq t_m(k) \tag{5.3}$$

where ξ_c and f_k are the server transmission rate and CPU cycles used for calculating data per bit, respectively.

As a result, the total required number of CPU clock cycles for calculating the given task k can be expressed as $\chi_k.f_k$. d, v, and T_h are the distance between cloud and edge, the speed of light, and the constant time of managing an incoming task, respectively. Online tasks can be transmitted to the edge network via wired or wireless links (BS), where the master node and edge server also reside. Handling and managing the offloaded tasks at the edge can be considered as local processing, so the latency is mainly correlated with the residual computing capacity to process the task. Latency caused by handling a task can be simply expressed as $T_h = \frac{\chi_k}{B_e \nu}$, where the discount factor is denoted as ν reflecting fluctuations of bandwidth at the edge ($0 < \nu < 1$). Accordingly, the offloading latency at the edge $t_e(k)$ can be calculated as below:

$$t_e(k) = \max_{j \in \eth(k)} \{ \frac{\chi_{k_j} f_{k_j}}{R^e_C(n_i)} + T_h \} \leq t_m(k) \tag{5.4}$$

PVs latency

Unlike the remote cloud or edge node that can be considered stable, PVs are possibly recognized as preemptible nodes due to their unpredictable mobility. Task replicas could be a promising approach to minimize service disruption. The master node could have sufficient time to migrate the current task replicas hosted to other worker nodes to maintain QoS regarding such a solution. In the normal state, replicas of a task can run under enabled load-balancing mode to improve availability, reliability, and performance efficiency.

Similar to [150], we leverage LTE-A for wireless links between the BS and PVs considering the orthogonal frequency-division multiple access (OFDMA) scheme. The parameter $d_{bs,p}$ expresses the distance between the BS and p^{th} PV while the path loss between them can be characterized by $d^{-\sigma}_{bs,p}$ and white Gaussian noise power N_0, where σ factor is the path loss exponent. The wireless channel is modeled as a frequency-flat block-fading Rayleigh fading channel for NLOS path, denoted as h. Hence, the downlink data rate of p^{th} PV is defined as:

$$\xi_p = B_p log_2(1 + \frac{P_{TX}.d^{-\sigma}_{bs,p}.|h^2|}{N_0 + I}) \tag{5.5}$$

where B_p, P_{TX} and I denote the channel bandwidth, transmission power of BS, and inter-cell interference, respectively. The offloading latency of PVs $t_p(k)$ can be then calculated as below:

$$t_p(k) = \max_{j \in \eth(k)} \{ \frac{\chi_{k_j}}{E[\xi_p]} + \frac{\chi_{k_j} f_{k_j}}{R_C^p(n_i)} + T_h \} \leq t_m(k) \tag{5.6}$$

Similar to [138], we present different types of online tasks, including latency-sensitive and latency-insensitive. The former can only be processed at the edge nodes (e.g., edge server, PVs) because of their closest proximity, whereas the latter can be handled at any worker nodes such as Cloud, Edge, and PVs. Thereupon, we calculate the mapping cost of a given task replica in the proposed collaborative framework. The mapping cost involves a sum of the total number of CPU clock cycles required to compute the task replica, memory, bandwidth, and energy consumption for handling replicas at PVs since cloud and edge produce great energy efficiency. Offloading cost at the Cloud can be formulated as below:

$$\Xi_{C_c}(k_j) = \frac{W_{C_c} \chi_{k_j} f_{k_j}}{C_c - \sum_{k' \in K_c} \sum_{j \leq |\eth(k')|} c(k'_j) + \delta} \tag{5.7}$$

$$\Xi_{M_c}(k_j) = \frac{W_{M_c} m(k_j)}{M_c - \sum_{k' \in K_c} \sum_{j \leq |\eth(k')|} m(k'_j) + \delta} \tag{5.8}$$

$$\Xi_{B_c}(k_j) = \frac{W_{B_c} \frac{\chi(k_j)}{t_m(k_j)}}{B_c - \sum_{k' \in K_c} \sum_{j \leq |\eth(k')|} \frac{\chi(k'_j)}{t_m(k'_j)} + \delta} \tag{5.9}$$

where δ denotes a small positive number to avoid dividing by zero. Total cost of offloading a task replica to the cloud is:

$$\Xi_{k_j}^c = \Xi_{C_c}(k_j) + \Xi_{M_c}(k_j) + \Xi_{B_c}(k_j) \tag{5.10}$$

As mentioned, if a given task is deployed at the edge, it can be acknowledged as a local processing, so the corresponding offloading cost at the Edge is calculated as below:

$$\Xi_{C_e}(k_j) = \frac{W_{C_e}\chi_{k_j}f_{k_j}}{C_e - \sum_{k' \in K_e}\sum_{j \leq |\eth(k')|} c(k'_j) + \delta} \tag{5.11}$$

$$\Xi_{M_e}(k_j) = \frac{W_{M_e}m(k_j)}{M_e - \sum_{k' \in K_e}\sum_{j \leq |\eth(k')|} m(k'_j) + \delta} \tag{5.12}$$

The total cost of offloading a task replica to the edge is:

$$\Xi^e_{k_j} = \Xi_{C_e}(k_j) + \Xi_{M_e}(k_j) \tag{5.13}$$

Likewise, the offloading cost of a task replica and its energy consumption at a PV can be calculated as below:

$$\Xi_{C_p}(k_j) = \frac{W_{C_p}\chi_{k_j}f_{k_j}}{C_p - \sum_{k' \in K_p}\sum_{j \leq |\eth(k')|} c(k'_j) + \delta} \tag{5.14}$$

$$\Xi_{M_p}(k_j) = \frac{W_{M_p}m(k_j)}{M_p - \sum_{k' \in K_p}\sum_{j \leq |\eth(k')|} m(k'_j) + \delta} \tag{5.15}$$

$$\Xi_{B_p}(k_j) = W_{B_p}\frac{\chi_{k_j}}{t_m(k_j)\xi_p;}, \forall k \in K_p \tag{5.16}$$

$$E_p(k_j) = \chi_{k_j}f_{k_j}e_p \tag{5.17}$$

where e_p denotes a coefficient, that can be obtained by:

$$e_p = \epsilon(R^p_C(n_i))^2 \tag{5.18}$$

where ϵ is an energy coefficient which indicates the trends in the ratio of energy consumption to the output level.

Eventually, total cost of offloading a task replica k_j to a PV is:

$$\Xi^p_{k_j} = \Xi_{C_p}(k_j) + \Xi_{M_p}(k_j) + \Xi_{B_p}(k_j) + \varsigma E_p(k_j); \tag{5.19}$$

where ς denotes energy cost coefficient.

PVs' Utility

By strongly encouraging owners of PVs to trade their idle computing resources when parking in parking lots, they can receive equivalent rewards by processing task replicas in their vehicles. Let denote φ^p as the rewards by accepting a task replica at a PV p. The utility can be computed as below:

$$\varpi^p = \varphi^p - \rho E_p(k_j) \tag{5.20}$$

where ρ is a coefficient of energy price, and φ^p is defined as:

$$\varphi^p = \mu r_p^c \chi_{k_j} f_{k_j} + r_p^m m(k_j) \tag{5.21}$$

where r_p^c and r_p^m represent unit prices of CPU and memory resources, respectively. It is directly observed that minimizing the offloading cost could boost profits earned by accepting the requested task replicas at PVs.

Variables:

$$\mathcal{A}_{k_j}^c = \begin{cases} 1, & k_j \text{ deployed on cloud, } \forall j \leq |\eth(k)|. \\ 0, & \text{otherwise.} \end{cases}$$

$$\mathcal{A}_{k_j}^e = \begin{cases} 1, & k_j \text{ deployed on edge, } \forall j \leq |\eth(k)|. \\ 0, & \text{otherwise.} \end{cases}$$

$$\mathcal{A}_{k_j}^p = \begin{cases} 1, & k_j \text{ deployed on a PV, } \forall j \leq |\eth(k)|. \\ 0, & \text{otherwise.} \end{cases}$$

Objective:

$$Minimize \sum_{j \leq |\eth k|} \Xi_{k_j}^c \mathcal{A}_{k_j}^c + \Xi_{k_j}^e \mathcal{A}_{k_j}^e + (\eta \Xi_{k_j}^p + (1-\eta)\frac{1}{\varpi^p})\mathcal{A}_{k_j}^p \tag{5.22}$$

$$w.r.t \qquad \mathcal{A}_{k_j}^c, \mathcal{A}_{k_j}^e, \mathcal{A}_{k_j}^p$$

Constraints:

$$\mathcal{A}_{k_j}^c + \mathcal{A}_{k_j}^e + \sum_{p \in N} \mathcal{A}_{k_j}^p = 1, j \leq |\eth(k)| \tag{5.23}$$

$$1 \leq \sum_{j \leq |\eth(k)|} \mathcal{A}^{p}_{k_j} \leq \alpha * |\eth(k)| \tag{5.24}$$

$$\sum_{j \leq |\eth(k)|} \mathcal{A}^{c|e|p}_{k_j} * c(k_j) \leq C_{c|e|p} \tag{5.25}$$

$$\sum_{j \leq |\eth(k)|} \mathcal{A}^{c|e|p}_{k_j} * m(k_j) \leq M_{c|e|p} \tag{5.26}$$

$$\sum_{j \leq |\eth(k)|} \mathcal{A}^{c}_{k_j} * b(k_j) \leq B_c \tag{5.27}$$

$$b(k_j) \leq \xi_{n_i}, \forall n_i \in N \tag{5.28}$$

$$\sum_{j \leq |\eth(k)|} \mathcal{A}^{c|e|p}_{k_j} * t_{c|e|p} \leq t_m(k) \tag{5.29}$$

Remarks:

- Function (5.22) is aimed at dual objectives: minimizing the offloading cost and maximizing PVs' rewards when task replicas are offloaded to PVs in which η is a damping factor within (0,1).
- Constraint (5.23) guarantees that a single task replica is merely handled on a worker node.
- Constraint (5.24) maintains that the proportion of task replicas assigned in each PV node cannot exceed 50% as PVs are possibly recognized as preemptible worker nodes because of their unpredictable mobility.
- Constraints (5.25),(5.26), (5.27), and (5.28) ensure the remaining resources of the worker nodes (e.g., Cloud, Edge, PVs) meet the stringent task requirements.
- Constraint (5.29) at last assures that the chosen nodes meet the latency constraint.

5.4 Simulation Setup

We have developed a discrete event simulator to assess the algorithms. Vehicles dynamically arrive and depart from parking lots with 50 or 100 free parking spots. Although the parking lot can be fully occupied, we assume that the parking capacity

only remains within the range of 50% up to a maximum of 85% in peak hours. The reason is that not all PVs are willing to trade their idle computing resources or are qualified for providing network services in reality (e.g., old-fashioned vehicles, insufficient computing capacity, very short parking availability, running errands). Also, it is observed that the parking duration of PVs is ranging from 8 to 240 minutes [135] or 30 to 120 minutes [149]. Following these behaviors, we can recognize that over 85% of PVs almost spent a maximum of three hours in a parking lot, and the probability of serviceability of PVs gains around 90% at 60 minutes [135]. The data are reliable due to the fact that they were abstracted from a real dataset from ACT Government Open Data Portal dataACT that uses the SmartParking app to collect 180295 parking records in the Manuka shopping precinct within 30 days. This data is huge and widely used in academia.

In this chapter, the accumulative parking duration of PVs follows the Poisson distribution with $\lambda = 3600$. Our simulation approximately runs for 8 hours (peak business working time), and the simulator will update the availability of PVs every 20 minutes. As aforementioned in previous sections, online task requests can be widely classified into latency-sensitive and latency-insensitive tasks. When a task's latency tolerance exceeds 20 ms, we considered it a latency-sensitive demand. Besides, energy coefficient ϵ, coefficient for energy price ρ and unit price for each CPU cycle σ are set to 10^{-24}, 0.003 and 2×10^{-9} [149], respectively. Details of simulation parameters can be found in Table 5.1.

5.5 M&M Heuristic Algorithm

BIP problems, that are computationally intractable to solve exactly, are widely known to be NP-hard. In [151], we propose a heuristic algorithm to obtain sub-optimal results by applying min-max algorithm (denoted as M&M) in aim of minimizing offloading costs while maximizing utility/rewards when given tasks are assigned to PVs. We compare M&M with three baseline algorithms, namely *Baseline_1, Baseline_2, Baseline_3*. Baseline_1 refers Kubernetes default scheduler with filtering and scoring procedures whereas Baseline_2 schedules online tasks with random policies which means that worker nodes are randomly selected for task offloading. Besides, Baseline_3 applies branch and bound selection policy as described in [152]. There are three

Table 5.1: Simulation Parameter Settings

Parameter	**Algorithms**
Maximum parking capacity	100
Total simulation time	30,000 seconds
Vehicle lifetime	[480-14400] seconds
C_c | M_c | B_c	50GHz | 1000MB | 1Gbps
C_e | M_e	20-25GHz | 500MB
$W_{\{C_c,M_c,B_c\}}$	750
$W_{\{C_e,M_e\}}$	250
$W_{\{C_p,M_p,B_p\}}$	10
Frequency	28 GHz
Channel Bandwidth B_p	10-50 MHz
P_{TX}	1.3W
N_0	3×10^{-13}
σ	2
CPU Parked Vehicles C_p	[1.5-2.0] GHz
Data rate ξ_p	[332-1065] Mbps
Input data size χ_k	[0.1 - 5.0] Mb
CPU cycles per bit f_k	1000-1500 cycles
Memory requests $m(k)$	[20-50] MB
Tolerable latency of tasks t_m	(0-100] ms
Arrival request rates	[10-120]
Request replications $\eth(k)$	[2 - 10]
r_p^c | r_p^m	10 | 100
Candidate selection proportion λ	0.25

performance metrics for evaluation including task average acceptance ratios (A/R), average costs and average accumulated utility/rewards. Details in our proposed algorithms are shown in Algorithm 6.

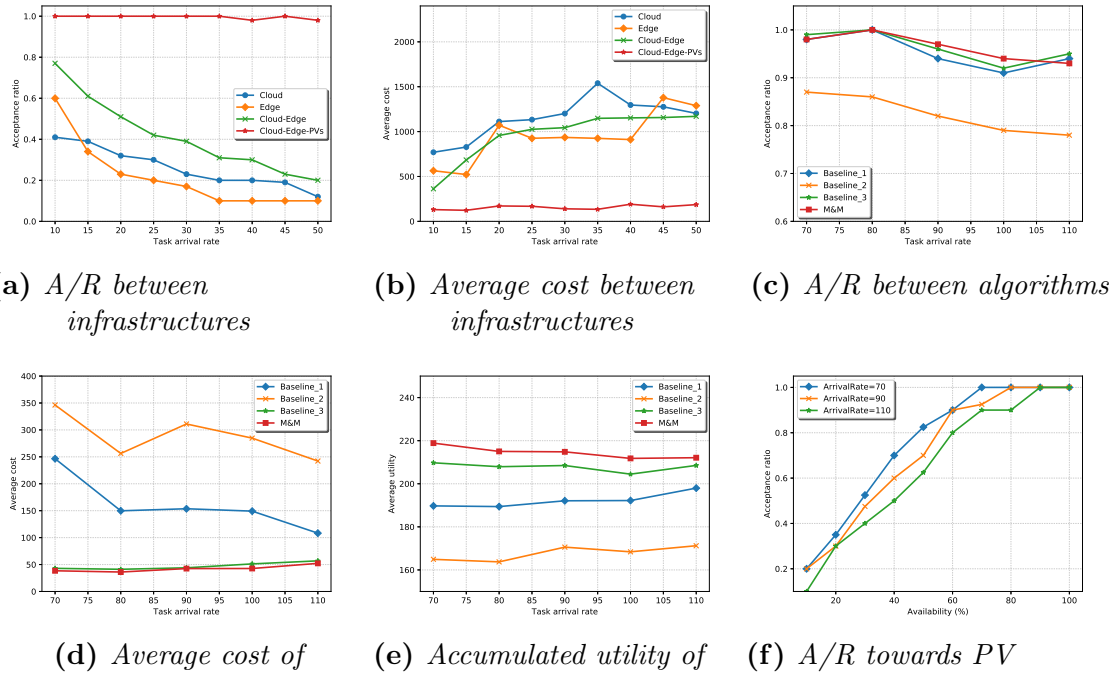
Algorithm 6 The M&M algorithm

function `GET_CANDIDATES`(k)
  `create an empty candidate array`
  for each replica of task k **do**
    for all $n \in N$ **do**[▷ ***including cloud, edge and PV nodes***]
      if ($CPU\_remaining \geq c(k)$ and $Memory\_remaining \geq m(k)$ **then**
        `add n to candidates`
  return candidates
function `NODE_SELECTION`(k)
  $utility \leftarrow zero$
  $candidates \leftarrow$ `GET_CANDIDATES(k)`
  sort descending all *candidates* based on their cost values detailed in Eq. (5.10), (5.13), (5.19).
  $new\_candidates \leftarrow \lambda$(`total_candidates`) [▷ λ***: candidate selection proportion***]
  for all `candidates` in `new candidates` **do**
    if candidate is a PV **then**
      calculate utility value based on Eq. (5.20)
  Select a candidate with highest utility value.
  return n

5.5.1 Performance Results

Figure 5.3a illustrates the effectiveness of our proposed architecture in which PVs are integrated into the ubiquitous network architecture to handle online task requests in peak hours. Cloud-Edge-PVs deployed M&M heuristic algorithm, that is extended the resource availability of the network, to increase the possibility of accepting the incoming requests. Specifically, the desired architecture improved the acceptance ratios from 23% up to 78% compared to common Cloud-Edge architecture at task arrival rates of 10 and 50, respectively. Cloud itself got lowest acceptance ratio when it was only able to execute delay-insensitive requests while Edge had a limited processing capacity.

By preferring to allocate task replicas to PVs, Cloud-Edge-PVs outperformed all compared platforms towards average offloading costs as illustrated in Fig. 5.3b. The proposed paradigm achieved at least 64% up to 85% better absolute average costs in a comparison with all compared infrastructures. The spikes in Fig. 5.3b might be caused by the stationary distribution of the online task requests. In contrast, as our proposed paradigm achieved significantly better performance than other architectures,



(a) *A/R between infrastructures* **(b)** *Average cost between infrastructures* **(c)** *A/R between algorithms*

(d) *Average cost of algorithms* **(e)** *Accumulated utility of algorithms* **(f)** *A/R towards PV availability*

Figure 5.3: *Performance evaluation*

we would like to stress its scheduling capability with other baselines at higher task arrival rates. In performance evaluation between compared algorithms, Baseline_2 attained worst performance in terms of acceptance ratios as well as averages costs due to its allocating strategies by selecting the worker nodes randomly without considering network compatibility. Baseline_1 based on filtering and scoring procedures accepted more tasks but was still lightly worse than Baseline_3 and M&M algorithms as shown in 5.3c. Regarding cost evaluation metric, M&M and Baseline_3 significantly performed better than others (Fig. 5.3d). It was not surprised since Baseline_3 was intentionally designed for minimizing the network cost. M&M seemed just lightly better than Baseline_3, but it could not compete our proposed algorithm in utility performance as illustrated in Fig. 5.3e. The reason is that we simultaneously considered both cost and utility at once, and the given tasks were most likely to be assigned to PVs which expectantly produced lower unit costs. Furthermore, we evaluated the availability of PVs on different arrival rates in respect of various acceptance ratios as depicted in Fig. 5.3f. It has been shown that when PV availability reached 85%, PVs are able to handle online heterogeneous tasks at all selected arrival rates successfully. In lower bound, we needed at least 60% availability of PVs remaining in parking lot to obtain

80% acceptance ratio.

5.6 Metaheuristic Genetic Algorithm

For solving the BIP problem, in [153] and [154], we propose a distributed and parallel GA-based algorithm that runs on several independent machines, denoted as V, to explore searching space. The design of our proposed parallel algorithm is depicted in Fig 5.4 in which $|V|$ is set to 16. The offloading procedures are successively working

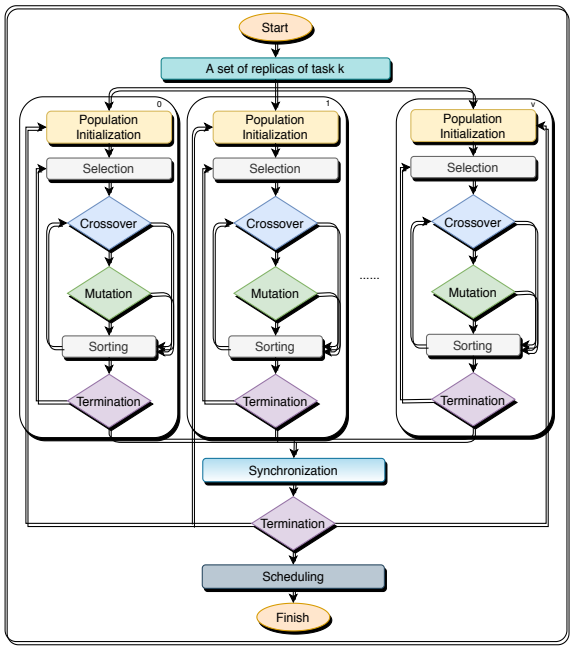


Figure 5.4: *Parallel operation scheme*

under a master machine (e.g., synchronization). Several distributed worker machines deploy the GA algorithm to discover as many feasible solutions for task offloading problems as possible. Each independent worker machine intentionally operates with several predefined iterations, hoping to select an ideal offloading solution amongst the distributed and parallel ones. The best solutions from the worker machines are then synchronized to determine the finalized solution for the online task offloading problem. In this paper, our proposed algorithm schedules multiple task replicas at once instead of sequentially mapping.

5.6.1 Genetic Representation and Selection

Chromosome: A chromosome $\mathcal{C}_f$ represents an offloading solution for a set of replicas of a given task request, selected from the available worker nodes that meet task resource requirements in random. Each gene within a chromosome is associated with a mapping solution for a single task replica, which can be represented as $g_f^j = \mathcal{A}_{k_j}^{c,f} \mathcal{A}_{k_j}^{e,f} \mathcal{A}_{k_j}^{1,f} \cdots \mathcal{A}_{k_j}^{|P|,f}$. Denote G as the number of genes, so $G = |\eth(k)|$. We begin the evolution process with M chromosomes, then the initial population can be generated as below:

$$\mathcal{P} = \begin{bmatrix} \mathcal{C}_1 \\ \mathcal{C}_2 \\ \vdots \\ \mathcal{C}_f \\ \vdots \\ \mathcal{C}_M \end{bmatrix} = \begin{bmatrix} g_1^1 & \cdots & g_1^j & \cdots & g_1^G \\ g_2^1 & \cdots & g_2^j & \cdots & g_2^G \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ g_f^1 & \cdots & g_f^j & \cdots & g_f^G \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ g_M^1 & \cdots & g_M^j & \cdots & g_M^G \end{bmatrix} \tag{5.30}$$

A chromosome, formed by G potential genes which have to pass a resource feasibility check, is identified as a feasible solution for a task offloading request with several replicas.

Selection: This operator mainly decides the chromosomes as parents for the prospective crossover operation. More than one pair of parental chromosomes can be selected from this operator. Moreover, to improve the parallelism degree, parents from the initial population stage are randomly appointed with a replacement. However, it is important to perceive that the mated children created in crossover operation could have better or even worse quality than their parents. Although there are literally many chromosome selection strategies for selecting the parents, the fitness-based proportionate designation, which is conceptually based on the cumulative sum of fitness-relative weights, is preferred in this operator.

Fitness Function: Our objectives are to minimize the offloading cost while maximizing task incentives when a set of replicas of a given task are offloaded to PVs. Fitness values evaluate the quality of the potential offloading solution, and higher

fitness values are preferable to being a good solution.

$$\mathcal{F}(k) = \sum_{j \in \partial k} \frac{1}{\Xi_k^c} \mathcal{A}_{k_j}^c + \frac{1}{\Xi_k^e} \mathcal{A}_{k_j}^e + ((1-\eta)\frac{1}{\Xi_k^p} + \eta\varphi_k^p)\mathcal{A}_{k_j}^p \tag{5.31}$$

5.6.2 The Processes of Evolution

In this paper, chromosomes are randomly chosen to become parents for generating an additional population. New evolved generations are generated as a result of crossover and mutation operations. To improve the diversity, such newly produced generations will be updated into the current population, which increases the possibility of approaching efficient offloading solutions.

Crossover: This operation is widely known as one of the most important stages in a typical GA algorithm to form new offspring by combining parental chromosomes. Denote $\mathcal{C}_s$ and $\mathcal{C}_r$ as two parental chromosomes with specific indexes s and r in the initial population. If j^c is a crossover point that is randomly chosen in any genes within N length, and then the corresponding descendant chromosomes are recognized as $\mathcal{C}_{(M+1)}$ and $\mathcal{C}_{(M+2)}$ respectively. By swapping parental genes starting from the crossover point $j^c + 1$ to the last one, new generations can be formed as illustrated below:

$$\mathcal{P} = \begin{bmatrix} \mathcal{C}_1 \\ \vdots \\ \mathcal{C}_s \\ \vdots \\ \mathcal{C}_r \\ \vdots \\ \mathcal{C}_M \\ \mathcal{C}_{M+1} \\ \mathcal{C}_{M+2} \end{bmatrix} = \begin{bmatrix} g_1^1 & \cdots & g_1^{j^c} & g_1^{j^c+1} & \cdots & g_1^G \\ \vdots & \ddots & \vdots & \ddots & \vdots & \\ g_s^1 & \cdots & g_s^{j^c} & g_s^{j^c+1} & \cdots & g_s^G \\ \vdots & \ddots & \vdots & \ddots & \vdots & \\ g_r^1 & \cdots & g_r^{j^c} & g_r^{j^c+1} & \cdots & g_r^G \\ \vdots & \ddots & \vdots & \ddots & \vdots & \\ g_M^1 & \cdots & g_M^{j^c} & g_M^{j^c+1} & \cdots & g_M^G \\ g_s^1 & \cdots & g_s^{j^c} & g_r^{j^c+1} & \cdots & g_r^G \\ g_r^1 & \cdots & g_r^{j^c} & g_s^{j^c+1} & \cdots & g_s^G \end{bmatrix} \tag{5.32}$$

Mutation: This stage usually deploys a modification on an individual parent to produce new offspring. A mutation operation is conducted to sample the broad searching space and improve the searching efficiency. The mutation is generally

considered an essential component of the GA processes, preventing solutions from falling into the local optima. A new gene can replace an existing gene of the under-processing chromosome selected at random to produce a new child. Such a new gene must explicitly meet the resource constraints to survive in the feasibility check. Denote j^m as the random mutation point while $g_{r'}^{j^m}$ is a new gene that replaces the existing one in $\mathcal{C}_{(M+1)}$. As a result, the solution of the mutation stage is $\mathcal{C'}_{(M+1)} = [g_s^1 \cdots g_{r'}^{j^m} \cdots g_s^G]$.

Algorithm 7 Proposed Intelligent GA-based Algorithm - EdgeGA

Input:
    A k^{th} online task with five tuples $\{c(k), m(k), b(k), t_m(k), \eth(k)\}$
Output:
    A set of worker nodes $n_i \in N$ where a number of task replicas will be assigned.
procedure TASK OFFLOADING
    Step 1: *Generate task candidates*
    function `GET_CANDIDATES` (k)
        `create an empty candidate array`
        for each online containerized task k **do**
            for all $n_i \in N$ **do**[▷ ***including cloud, edge and PV nodes***]
                if ($CPU\_remaining \geq c(k)$, $Memory\_remaining \geq m(k)$,$Bandwidth\_available \geq b(k)$, and $t_{n_i} \leq t_m(k)$ **then**
                    `add` n_i `to candidate array`
        return candidate array
        if (no available worker nodes found) **then**
            return *Task offloading failed*
    Step 2: *Implement* **Genetic Algorithm** *in distributed and parallel machines as shown in* ***Algorithm 7***
    Step 3: *Synchronize achieved incumbents among parallel machines*
    Step 4: *Select the global incumbent solution based on sum of fitness values (E.q. 5.31)*
    return a set of $n_i \in N$
    Update substrate network information

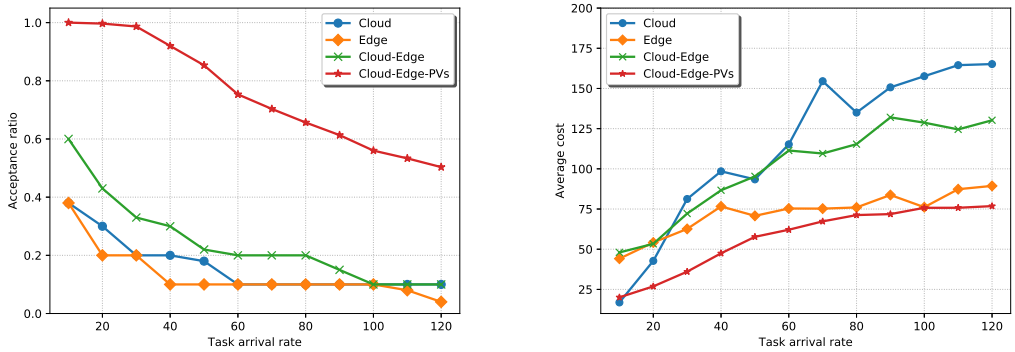
Algorithm 8 Genetic Algorithm at each paralleled machine

Input:
    Task candidates
    Set parameters
Output:
    Output the best offloading solution
procedure GA OPERATIONS
    Initial population *Generate M chromosomes, each chromosome has G genes, $G = |\eth(k)|$. Each gene is a solution of a task replica, which is selected from the task candidates in random* $g_f^j = \mathcal{A}_{k_j}^{c,f}\mathcal{A}_{k_j}^{e,f}\mathcal{A}_{k_j}^{1,f}\cdots\mathcal{A}_{k_j}^{|P|,f}$.
    while $(i \leq maxIterations)$ **do**
        Selection *Select two random chromosomes as parents*
        Crossover *Swapping parental genes starting from a random crossover point to the last one.*
        Mutation *A new child is generated by replacing an existing gene of the parental chromosome with a new one in random.*
    Output the best solution known as incumbent based on fitness values (E.q. 5.31)

5.6.3 Terminations and Synchronization

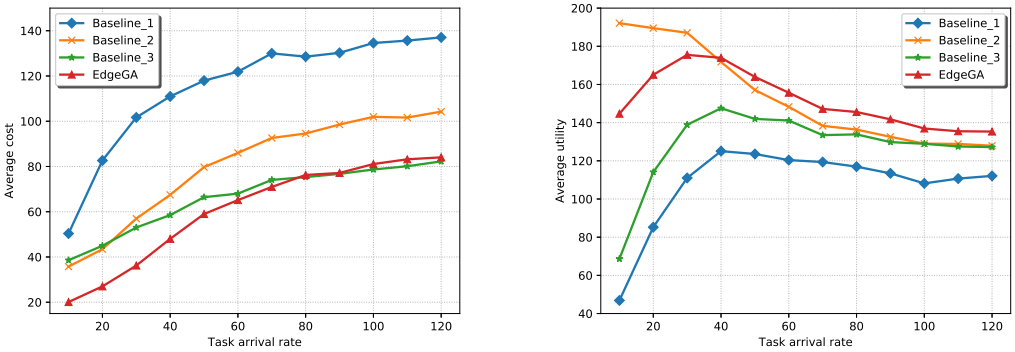
A parallel operation commonly includes several concurrent processes where each process may finish its assigned job at a different time. However, waiting for all tasks to accomplish their assignments is troublesome because one or several tasks can take too much time to be successfully processed (e.g., deadlock). Similar to Section 3.4.5, if there is no better solution achieved within t times, the master procedure will terminate the GA algorithm running at worker machines to reduce total execution time. Feasible solutions of GA algorithm gathered from the corresponding worker machines can be synchronized to determine the best offloading solution based on fitness function values. If the best solution is found and accepted for the task offloading request, its task replicas are then assigned onto the corresponding worker nodes. Substrate network consequently updates the remaining computing resources to accomplish the offloading procedures. Details on our proposed distributed and parallel GA-based algorithm for online task offloading problem can be found in Algorithm 7.

We measure the execution time of the proposed framework on two different modes: sequential and parallel. In the former, the time complexity increases linearly since the total time equals the sum of the execution time of each worker machine. In



(a) **(b)**

Figure 5.5: (a) Acceptance ratio (b) Average costs between architectures



(a) **(b)**

Figure 5.6: (a) Average offloading cost (b) Average utility

contrast, the latter's total execution time is counted by the worker machine that accomplishes its assigned task at the latest. Cramer-Chernoff method and Jensen's inequality can approximate the total execution time in parallel mode as detailed in 3.3.2. Accordingly, the proposed distributed and parallel framework can rapidly improve the time complexity from linear to logarithmic scale.

5.6.4 Compared Algorithms

Similar to 5.5, we evaluate the efficiency of our proposed GA-based algorithm by comparing it with some heuristic algorithms, including Baseline_1, Baseline_2, and

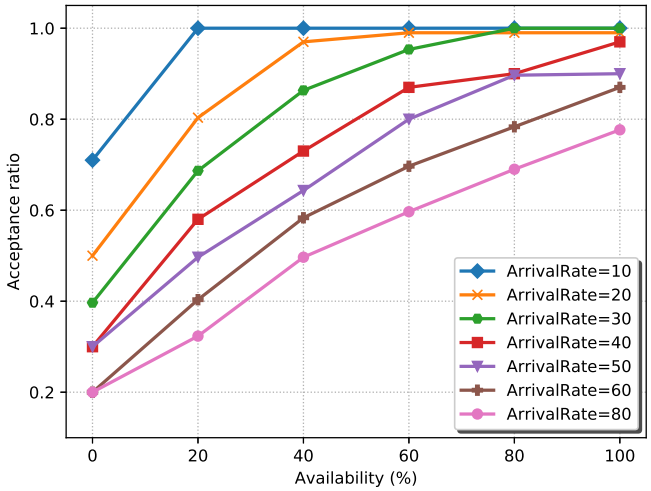


Figure 5.7: Acceptance ratio towards PV availability

Baseline_3. Baseline_1 prefers Kubernetes default scheduler with filtering and scoring procedures, while Baseline_2 schedules online task replicas by determining worker nodes for task offloading randomly. Additionally, Baseline_3 implements a branch and bound selection policy to address incoming tasks [152]. Our proposed algorithm generally allows multiple task replicas to be handled onto the same worker node. Still, the maximum proportion of the number of task replicas placed on this node cannot exceed 50% (except cloud and edge nodes) to guarantee service reliability. SPs can simply adjust this setup. Different metrics for performance evaluation consist of average task acceptance ratios, average offloading costs, and average accumulated utility. Furthermore, we compare our algorithm on different sizes, including 50 and 100 spots equivalent to small and medium parking lots.

5.6.5 Performance Results

Towards the small size of the parking lot, simulation results are illustrated in Fig. 5.5, 5.6 and 5.7 while those of medium size of the parking lot is depicted in Fig. 5.8 and 5.9. Eventually, Fig. 5.10 shows an execution time comparison of our proposed GA-based algorithm between different sizes.

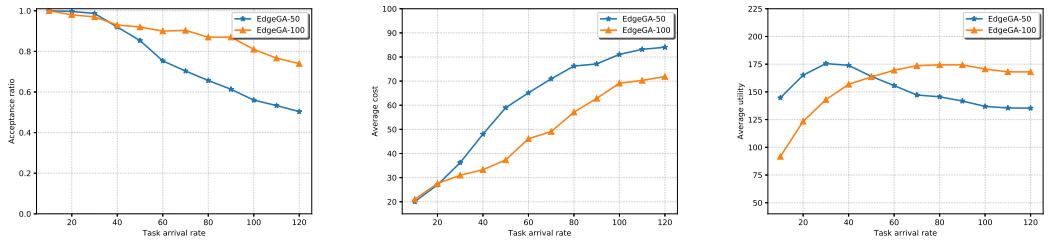


Figure 5.8: EdgePV's evaluation on different sizes: (a) Acceptance ratio (b) Average cost (c) Average utility

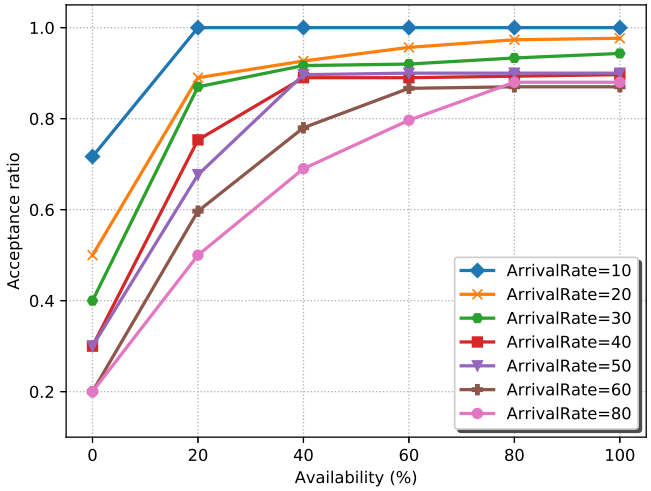


Figure 5.9: Acceptance ratio towards PV availability with 100 PVs

Fig. 5.5a shows that the collaborative Cloud-Edge-PVs framework extended the network resource availability, improving the average acceptance ratio of online task requests by more than 40% compared to Cloud-Edge and Cloud infrastructures task arrival rate of 120. Cloud or edge paradigm achieved the lowest acceptance ratio because of their limited computing capacity during peak hours. In contrast, the proposed Cloud-Edge-PVs framework produced the lowest average cost compared to all compared platforms, as depicted in Fig. 5.5b.

In evaluation amongst compared algorithms in Fig. 5.6a, Baseline_1 performed worst regarding average offloading cost due to its allocating strategies with filtering and scoring procedures. Baseline_2 relied on a random strategy for choosing the worker nodes that performed better than Baseline_1. Towards baseline algorithms,

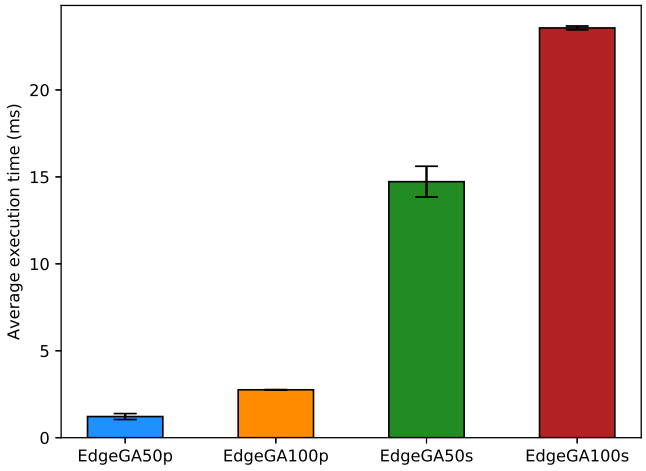


Figure 5.10: Average execution time with sequential and parallel operations

Baseline_3 was primarily originated in reducing offloading cost, so its performance was very comparative to our proposed algorithm. As illustrated in Fig. 5.6a, our proposed EdgeGA still performed better than Baseline_3 until the arrival rate of 80 and appeared to be lightly the same afterward, since online tasks would be most likely allocated to PVs, which is expected to produce lower offloading cost. In the utility aspect, EdgeGA outperformed all compared algorithms following Baseline_2,Baseline_3, and Baseline_1, respectively, as shown in Fig. 5.6b. The reason is that EdgeGA simultaneously considered both offloading costs and utility-driven by an efficient fitness function (5.31).

Additionally, we evaluated the availability of PVs towards the acceptance ratios on various task arrival rates as demonstrated in Figure 5.7. Each arrival rate had different preferable PV availability. For example, arrival rates (10, 20, 30, 40, 50) required 60% availability of PVs to exceed 80% acceptance ratios while arrival rates of 60 and 80 needed to reach 80% and 100% to obtain the same result, respectively.

Furthermore, we evaluated the proposed algorithm on the medium size of the parking lot with 100 spots (denoted as EdgeGA-100). We then compared the one on the smaller size that has been already done in the previous part (denoted as EdgeGA-50). As depicted in Fig. 5.8a and 5.8c, the acceptance ratio and average utility were both increased up to 24% at the arrival rate of 120 while the size of the parking lot was doubled. At the same rate, the offloading cost was also improved

by more than 16% as shown in Fig. 5.8b. Through these evaluation results, IPs can decide to select small or medium sizes of parking lots to achieve a trade-off between the received revenues and network cost.

Similarly, we quantified the acceptance ratios towards higher PV availability. To achieve an 80% acceptance ratio for all arrival rates, we merely needed to reach 60% of PV availability as opposed to 100% with a smaller size. Arrival rates of $10, 20$ and 30 rapidly obtained 80% of acceptance ratio with PV availability less than 20% while arrival rates of $40, 50$ and 60 required 40% to gain the same. Eventually, an arrival rate of 80 acquired more than 20% to obtain the same result. This information is vital for the network planners to achieve desired Key Performance Indicators (KPIs) by adopting appropriate strategies. For instance, SPs may increase user incentives to attract more PVs to join the network, offload to another cluster or expand edge server capacity.

Furthermore, our proposed GA-based algorithm successfully processed a given task in average $1.217ms$ compared to $14.725ms$ in sequential GA operation as depicted in Fig. 5.10. By expanding the parking lot to 100 spots, it requires $2.756ms$ and $23.567ms$ to finish the same job with parallel (_p) and sequential (_s) modes respectively. This superior execution-time performance is attained by deploying the parallel scheme for the GA algorithm as proposed in Fig. 5.4. This execution time is very competitive. It somehow lifts the curse of high computation time when running the GA algorithm.

5.7 Chapter Summary

This chapter has presented the collaborative framework in which PVs are potentially considered an extension for the existing cloud-edge computing infrastructure to handle online container-based tasks in peak hours. We have devised Kubernetes, a container orchestrator, deployed at edge server as the master node, while the remote cloud, edge itself, and PVs perform as worker nodes. Extensive experiments demonstrated that our proposed collaborative framework extends the computing capability of the existing network infrastructure and improves the offloading cost for at least 40% by exploiting powerful on-board hardware of PVs efficiently and brings a flexible, agile, and reliable framework for online task offloading problems. Moreover, we proposed a heuristic algorithm, namely M&M, and GA-based algorithms compared with several baseline algorithms on several performance metrics. Our proposed algorithm outperformed

all competitors regarding task acceptance ratios, offloading cost, and accumulative rewards. Owners of PVs can earn profits by trading idle computing resources during parking in parking lots. We also compare our proposed GA-based algorithms on different sizes of generic parking lots. Furthermore, we quantify the relationship between PV availability and task acceptance ratios on several arrival rates. This information is indeed crucial to SPs for determining appropriate offloading strategies to maximize network revenues. Thanks to the distributed and parallel operation scheme, our proposed GA-based algorithm significantly reduces the execution time compared to sequential operation. Towards the scalability of Kubetnetes platform, it essentially follows the cluster concept where the smaller logical component is zone. We can deploy multiple controllers to handle the scalability issue. Each controller can manage one or more zones with an individual allocation algorithm. If the current capacity of a cluster is exceeded, another cluster can be exploited to handle the exceeded amount of vehicles.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this dissertation, we have investigated the efficiency of several distributed and parallel GA-based algorithms on different resource allocation problems. Metaheuristic GA algorithm could approach efficient mapping solutions while reducing GA's running time by adopting the distributed and parallel operation implementations. Our proposed solutions not only provide a fast resource allocation approach that can be applicable to online embedding applications. but also produce high performance efficiency.

NV is ubiquitously an essential attribute to enable success of the figure virtualized networks (e.g. 5G and beyond, smart IoTs). VNE is the main challenge in NV that allows multiple heterogeneous VNs to simultaneously coexist on the top of a shared substrate infrastructure. Many VNE solutions have been proposed over the past decade, but most of them only focus on the node mapping, leaving link mapping for the shortest path methods or MCF mechanism. In first part of this dissertation, we propose an intelligent GA-based algorithm implemented in a distributed and parallel manner for VLiM stage [13]. Its improvements that are consisted of a multi-objective fitness function [128] and a novel crossover operator for GA algorithm's procedures [129] are also introduced. These VNE solutions are based on a very simple VNoM mechanism, so we advise a node ranking method for virtual node embedding that is cooperated with our intelligent VNE framework for link mapping stage to improve VNE solutions in [155]. A new mutation operation that is associated with a path-based gene modification is also proposed. Extensive simulation results show that our proposals not only perform better than many VNE algorithms in performance, but also exceed the popular greedy node mapping with the shortest path method for link

mapping stage, which has been perceived as the most popular and fastest unsplittable VNE algorithm, in speed. Our framework can be considered as an iconic proposal for solving resource allocation with respect to two critical aspects: great performance and rapid speed.

Although, our proposed algorithms achieved better embedding performance than several heuristic VNE algorithms, a lack of coordination between VNoM and VLiM stages could result in low embedding results due to the fact that separating VNE stages makes VLiM have no chance to remap node mappings of a given VNR when the link mapping algorithm could not find any feasible path for the corresponding virtual link request in respect to the node mapping solutions in VNoM stage. Then, the corresponding VNR is certainly rejected. This coupling can facilitate the algorithmic deployment, but it can result in low average acceptance ratio since the most common failures of VNE are derived from infeasible link mappings. In the second part of this dissertation, we propose a join node and link algorithm for mapping VNRs based on GA algorithm. GA attempts to seek for potential solutions for all virtual nodes in a given VNR, then a path ranking method is deployed to find link mappings for corresponding virtual links. Some virtual link requests are possibly failed to be mapped since there are no feasible paths due to network congestion, current node mappings must be reconsidered with minimal node and link mappings changed. Consequently, we propose a novel conciliation algorithm to address this problem. GA algorithms run on a distributed and parallel scheme that exploits a set of distributed and parallel machines in order to reduce the operation time. A brief comparison towards execution time between sequential and parallel manners is also provided. Our extensive evaluation shown that the proposed algorithm outperformed state-of-the-art VNE algorithms in all performance metrics we adopted. It is observed that the achieved results (e.g., average acceptance ratio is greater than 92% and 72% in average at traffic loads of 40 and 80) are indeed a real challenge for any future VNE algorithm including RL algorithms with the same parameter settings, we suppose.

As cited in 1.2, we would like to leverage our proposed solution to solve the resource allocation problem in another different research field. In fact, a recent analytical research has pointed out that almost all vehicles spend over 95% of their time in parking lots where their powerful computing resources are wasted. In last part of this dissertation, we propose a novel collaborative computing paradigm that efficiently offloads online heterogeneous computation tasks to parked vehicles (PVs) during

peak hours. A container orchestration based on Kubernetes is advocated to integrate into the existing infrastructure due to its cutting-edge features such as auto-healing, load-balancing, and security. We formulate the offloading problem analytically and present two efficient heuristic algorithms to address dynamic online demands. The first algorithm is M&M which is aimed at minimizing offloading cost while maximizing utility/rewards when given tasks assigned to PVs. Extensive evaluation demonstrates that the M&M-assisted paradigm improved task arrival ratios from 23% up to 78% compared to the most dominant Cloud-Edge architecture while reducing average offloading cost for more than 64%. M&M algorithm also outperformed three baseline heuristic algorithms in all compared performance metrics we adopted. However, M&M does not completely consider where the task replicas are allocated, which means that all task replicas are able to be placed in the same single worker node. Occasionally, if the selected node accidentally encounters an unexpected failure (e.g., power outage, sudden leaving of vehicles), containerized services running on it would be disrupted. To solve such vulnerability and maintain service continuity as well as reliability, an upper bound of the proportion of task replicas running on a worker node is taken into account. We then introduce a metaheuristic GA algorithm that is guided by the efficient fitness function to solve the problem of time complexity and scalability of the BIP problem. The metaheuristic algorithm was examined on different sizes of parking lots to prove its efficiency. The performance evaluation showed that the acceptance ratio and average utility are both enhanced up to 24% at the highest arrival rate while the size is doubled. At the same rate, the offloading cost was also improved for more than 16%. From these evaluation results, IPs can determine the trade-off between the sizes of parking lots and network metrics. Moreover, we proposed a distributed and parallel scheme as described in Section 2.4 running GA algorithm to reduce the execution time. Additionally, owners of PVs can be beneficial by sharing their idle vehicle resources through received incentives in both algorithms. We also simulate the model with the random mobility of PVs under random and dynamic task arrivals. Furthermore, we evaluated the availability of PVs on different arrival rates in respect to various acceptance ratios. This information is vital for network planners to achieve desired KPIs by adopting appropriate strategies. Our preliminary results indicate that our proposed framework and algorithms are promising to address online task offloading problems, deal with the waste of computational resources of PVs and profit owners of PVs by selling their idle resources during parking. Chapter 5 and other

chapters are all related to virtual resource allocation. Chapter 5 vitally brings our proposed approach from the core network down to the access network, specifically edge network with different objectives, which demonstrates the broad applicability of our approach. Combining all chapters in a single dissertation makes it more complete.

6.2 Future Work

Current resource allocation approaches have not considered energy objective which can be formulated as non-linear optimization problems. In that case, our GA-based approaches will be applicable since GA algorithm can be applied to solve linear or nonlinear problems.

In addition, VNE is a generic resource allocation technique in NV where virtual optical network embedding (VONE) that addresses NV in optical infrastructure can be considered as a special case of the VNE problems due to a mutual domain. However, virtual links in VONE are mapped onto paths that are formed by several optical links, so wavelength and spectrum allocations need to be taken into account with some specific constraints.

In fact, the conciliation strategy of our joint node-link approaches simply carries out the random remapping instead of attempting all possible combinations and then selecting the best one. Hence, there are some potentials to further explore this remapping strategy.

Furthermore, the GA's operations in our joint node and link mapping approaches are conventional, so that we believe that they still have more rooms to be improved. For example, elastic crossover in Section 3.6.2 or multi-point mutation operation can be deployed in these operations. To enhance the solution space exploration that gives more chances to achieve better parents, the order of mutation operation could be also rearranged in the initialization population generation in future projects.

On the other hand, we are keen on developing a prototype of our proposed collaborative computation framework that can run on an actual Kubernetes platform, which brings our research close to real life. We also do not ignore the possibility of comparing our proposed GA algorithm with RL/DRL algorithms in this topic.

List of References

[1] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 1888–1906, Fourth Quarter 2013, doi: 10.1109/SURV.2013.013013.00155.

[2] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the internet impasse through virtualization," *IEEE Computer*, vol. 38, p. 34–41, Apr. 2005, doi: 10.1109/MC.2005.136.

[3] M. Handley, "Why the internet only just works," *BT Technology Journal*, vol. 24, pp. 119–129, Jul 2006, doi: 10.1007/s10550-006-0084-z.

[4] M. F. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani, "Data center network virtualization: A survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 2, pp. 909–928, Second Quarter 2013, doi: 10.1109/SURV.2012.090512.00043.

[5] K. Kirkpatrick, "Software-defined networking," *Communications of the ACM*, vol. 56, p. 16–19, Sept. 2013, doi: 10.1145/2500468.2500473.

[6] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, Feb. 2015, doi: 10.1109/MCOM.2015.7045396.

[7] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. K. Soong, and J. C. Zhang, "What will 5G be?," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 6, pp. 1065–1082, 2014.

[8] I. Ishaq, J. Hoebeke, I. Moerman, and P. Demeester, "Internet of things virtual networks: Bringing network virtualization to resource-constrained devices," in *2012 IEEE International Conference on Green Computing and Communications*, Besancon, 2012, pp. 293-300, doi: 10.1109/GreenCom.2012.152.

[9] J. Gil Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, 2016.

[10] T. A. Q. Pham, J.-M. Sanner, C. Morin, and Y. Hadjadj-Aoul, "Virtual network function–forwarding graph embedding: A genetic algorithm approach," *International Journal of Communication Systems*, vol. 33, no. 10, p. e4098, 2020. e4098 0.1002/dac.4098.

[11] B. Addis, G. Carello, and M. Gao, "On a virtual network functions placement and routing problem: Some properties and a comparison of two formulations," *Networks*, vol. 75, no. 2, pp. 158–182, 2020.

[12] M. Chowdhury, M. R. Rahman, and R. Boutaba, "Vineyard: Virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM Transactions on Networking*, vol. 20, pp. 206–219, Feb 2012, doi: 10.1109/TNET.2011.2159308.

[13] K. Nguyen and C. Huang, "An intelligent parallel algorithm for online virtual network embedding," in *2019 International Conference on Computer, Information and Telecommunication Systems (CITS)*, Beijing, China, 2019, pp. 1-5, doi: 10.1109/CITS.2019.8862072.

[14] Y. He, Z. Zhang, F. R. Yu, N. Zhao, H. Yin, V. C. M. Leung, and Y. Zhang, "Deep-reinforcement-learning-based optimization for cache-enabled opportunistic interference alignment wireless networks," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 11, pp. 10433–10445, 2017.

[15] Y. Wei, F. R. Yu, M. Song, and Z. Han, "User scheduling and resource allocation in hetnets with hybrid energy supply: An actor-critic reinforcement learning approach," *IEEE Transactions on Wireless Communications*, vol. 17, no. 1, pp. 680–692, 2018.

[16] C. Qiu, F. R. Yu, H. Yao, C. Jiang, F. Xu, and C. Zhao, "Blockchain-based software-defined industrial internet of things: A dueling deep Q -learning approach," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4627–4639, 2019.

[17] Y. He, C. Liang, F. R. Yu, and Z. Han, "Trust-based social networks with computing, caching and communications: A deep reinforcement learning approach," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 1, pp. 66–79, 2020.

[18] N. Zhao, H. Wu, F. R. Yu, L. Wang, W. Zhang, and V. C. Leung, "Deep reinforcement learning-based latency minimization in edge intelligence over vehicular networks," *IEEE Internet of Things Journal*, pp. 1–1, 2021.

[19] H. Yao, X. Chen, M. Li, P. Zhang, and L. Wang, "A novel reinforcement learning algorithm for virtual network embedding," *Neurocomputing*, vol. 284, pp. 1–9, 2018.

[20] H. Yao, S. Ma, J. Wang, P. Zhang, C. Jiang, and S. Guo, "A continuous-decision virtual network embedding scheme relying on reinforcement learning," *IEEE Transactions on Network and Service Management*, vol. 17, no. 2, pp. 864–875, June 2020, doi: 10.1109/TNSM.2020.2971543.

[21] D. Andreoletti, T. Velichkova, G. Verticale, M. Tornatore, and S. Giordano, "A privacy-preserving reinforcement learning algorithm for multi-domain virtual network embedding," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2291–2304, Dec. 2020, doi: 10.1109/TNSM.2020.3022278.

[22] P. Zhang, C. Wang, C. Jiang, and A. Benslimane, "Security-aware virtual network embedding algorithm based on reinforcement learning," *IEEE Transactions on Network Science and Engineering*, pp. 1–1, 2020, doi: 10.1109/TNSE.2020.2995863.

[23] Z. Yan, J. Ge, Y. Wu, H. Zheng, L. Li, and T. Li, "Automatic virtual network embedding based on deep reinforcement learning," in *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, Zhangjiajie, China, 2019, pp. 625-631, doi: 10.1109/HPCC/SmartCity/DSS.2019.00095.

[24] Z. Yan, J. Ge, Y. Wu, L. Li, and T. Li, "Automatic virtual network embedding: A deep reinforcement learning approach with graph convolutional networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1040–1057, June 2020, doi: 10.1109/JSAC.2020.2986662.

[25] P. Zhang, C. Wang, G. S. Aujla, and X. Pang, "A node probability-based reinforcement learning framework for virtual network embedding," in *2020 IEEE 21st International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, Cork, Ireland, 2020, pp. 421-426, doi: 10.1109/WoWMoM49955.2020.00077.

[26] R. Mijumbi, J. Gorricho, J. Serrat, M. Claeys, F. De Turck, and S. Latré, "Design and evaluation of learning algorithms for dynamic resource management in virtual networks," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, Krakow, 2014, pp. 1-9, doi: 10.1109/NOMS.2014.6838258.

[27] R. Mijumbi, J. Gorricho, J. Serrat, M. Claeys, J. Famaey, and F. De Turck, "Neural network-based autonomous allocation of resources in virtual networks," in *2014 European Conference on Networks and Communications (EuCNC)*, Bologna, 2014, pp. 1-6, doi: 10.1109/EuCNC.2014.6882668.

[28] H. Zhang, X. Zheng, J. Tian, and Q. Xue, "A virtual network embedding algorithm based on rbf neural network," in *2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, vol. 1, Guangzhou, 2017, pp. 393-396, doi: 10.1109/CSE-EUC.2017.77.

[29] M. He, L. Zhuang, S. Tian, G. Wang, and K. Zhang, "Multi-objective virtual network embedding algorithm based on q-learning and curiosity-driven," *EURASIP Journal on Wireless Communications and Networking*, vol. 2018, p. 150, Jun 2018, doi: 10.1186/s13638-018-1170-x.

[30] S. Wang, J. Bi, J. Wu, A. V. Vasilakos, and Q. Fan, "Vne-td: A virtual network embedding algorithm based on temporal-difference learning," *Elsevier Computer Networks*, vol. 161, pp. 251 – 263, 2019, doi: 10.1016/j.comnet.2019.05.004.

[31] M. Dolati, S. B. Hassanpour, M. Ghaderi, and A. Khonsari, "Deepvine: Virtual network embedding with deep reinforcement learning," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM*

WKSHPS), pp. 879–885, Paris, France, 2019, pp. 879-885, doi: 10.1109/INFCOMW.2019.8845171.

[32] A. Blenk, P. Kalmbach, P. van der Smagt, and W. Kellerer, "Boost online virtual network embedding: Using neural networks for admission control," in *2016 12th International Conference on Network and Service Management (CNSM)*, Montreal, QC, 2016, pp. 10-18, doi: 10.1109/CNSM.2016.7818395.

[33] A. Rkhami, T. A. Quang Pham, Y. Hadjadj-Aoul, A. Outtagarts, and G. Rubino, "On the use of graph neural networks for virtual network embedding," in *2020 International Symposium on Networks, Computers and Communications (ISNCC)*, pp. 1–6, Montreal, QC, 2020, pp. 1-6, doi: 10.1109/ISNCC49221.2020.9297270.

[34] H. Mühlenbein, "Parallel genetic algorithms in combinational optimization," in *Computer Science and Operations Research* (O. Balci, R. Sharda, and S. A. Zenios, eds.), pp. 441 – 453, Amsterdam: Pergamon, 1992, doi: 10.1016/B978-0-08-040806-4.50034-4.

[35] G. Yang, "The application of mapreduce in the cloud computing," in *2011 2nd International Symposium on Intelligence Information Processing and Trusted Computing*, pp. 154–156, Hubei, 2011, pp. 154-156, doi: 10.1109/IPTC.2011.46.

[36] Q. Lu, K. Nguyen, and C. Huang, "Distributed parallel algorithms for online virtual network embedding applications," *Wiley International Journal of Communication Systems*, 24 Jan 2020, pp. e4325,doi: 10.1002/dac.4325.

[37] Q. Lu, K. Nguyen, and C. Huang, "A novel one-stage distributed parallel embedding for virtualized network environment," in *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Toronto, ON, Oct. 2020, pp. 395-400, doi: 10.1109/SMC42975.2020.9282829.

[38] Q. Lu, K. Nguyen, and C. Huang, "Gaone: A novel approach for online one-stage virtual functions embedding," *Journal of Networking and Network Applications*, 2021.

[39] H. Cao, S. Wu, Y. Hu, Y. Liu, and L. Yang, "A survey of embedding algorithm for virtual network embedding," *China Communications*, vol. 16, no. 12, pp. 1–33, Dec. 2019, doi: 10.23919/JCC.2019.12.001.

[40] H. Cao, H. Hu, Z. Qu, and L. Yang, "Heuristic solutions of virtual network embedding: A survey," *China Communications*, vol. 15, no. 3, pp. 186–219, March 2018, doi: 10.1109/CC.2018.8332001.

[41] D. Soldani, K. Pentikousis, R. Tafazolli, and D. Franceschini, "5g networks: End-to-end architecture and infrastructure [guest editorial]," *IEEE Communications Magazine*, vol. 52, no. 11, pp. 62–64, Nov. 2014, doi: 10.1109/MCOM.2014.6957144.

[42] I. Khalil, A. Khreishah, and M. Azeem, "Cloud computing security: A survey," *MDPI Computers*, vol. 3, p. 1–35, Feb 2014, doi: 10.3390/computers3010001.

[43] 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*, vol. 38, pp. 17–29, Mar. 2008, doi: 10.1145/1355734.1355737.

[44] D. Eppstein, "Finding the k shortest paths," *SIAM Journal on Computing*, vol. 28, p. 652–673, Feb. 1999, doi: 10.1137/S0097539795290477.

[45] C. Barnhart, N. Krishnan, and P. H. Vance, *Multicommodity flow problems*. Boston, MA: Springer US, 2001, ISBN: 978-0-306-48332-5.

[46] D. Bertsimas and J. Tsitsiklis, *Introduction to Linear Optimization*. Athena Scientific, 1st ed., 1997, ISBN:978-1-886529-19-9.

[47] G. S. Paschos, M. A. Abdullah, and S. Vassilaras, "Network slicing with splittable flows is hard," in *2018 IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, Bologna, 2018, pp. 1788-1793, doi: 10.1109/PIMRC.2018.8580968.

[48] N. M. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *IEEE INFOCOM 2009*, Rio De Janeiro, Brazil, 2009, pp. 783-791, doi: 10.1109/INFCOM.2009.5061987.

[49] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, and J. Wang, "Virtual network embedding through topology-aware node ranking," *SIGCOMM Computer Communication Review*, vol. 41, p. 38–47, Apr. 2011, doi: 10.1145/1971162.1971168.

[50] F. Samuel, M. Chowdhury, and R. Boutaba, "Polyvine: policy-based virtual network embedding across multiple domains," *Journal of Internet Services and Applications*, vol. 4, p. 6, Mar 2013, doi: 10.1186/1869-0238-4-6.

[51] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Computer Networks*, vol. 54, no. 5, pp. 862 – 876, 2010, doi: 10.1016/j.comnet.2009.10.017.

[52] L. Wolsey, *Integer Programming, Second Edition*. John Wiley & Sons, 2nd ed. , Sep. 2020, 10.1002/9781119606475.

[53] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd ed., Jul. 2009, ISBN: 0262033844.

[54] M. Melo, S. Sargento, U. Killat, A. Timm-Giel, and J. Carapinha, "Optimal virtual network embedding: Node-link formulation," *IEEE Transactions on Network and Service Management*, vol. 10, no. 4, pp. 356–368, 2013, doi: 10.1109/TNSM.2013.092813.130397.

[55] R. Mijumbi, J. Serrat, J. Gorricho, and R. Boutaba, "A path generation approach to embedding of virtual networks," *IEEE Transactions on Network and Service Management*, vol. 12, pp. 334–348, Sep. 2015, doi: 10.1109/TNSM.2015.2459073.

[56] Q. Hu, Y. Wang, and X. Cao, "Resolve the virtual network embedding problem: A column generation approach," in *2013 Proceedings IEEE INFOCOM*, pp. 410–414, 2013, doi: 10.1109/INFCOM.2013.6566805.

[57] A. Jarray and A. Karmouch, "Decomposition approaches for virtual network embedding with one-shot node and link mapping," *IEEE/ACM Transactions on Networking*, vol. 23, no. 3, pp. 1012–1025, 2015, doi: 10.1109/TNET.2014.2312928.

[58] C. Huang and J. Zhu, "Modeling service applications for optimal parallel embedding," *IEEE Transactions on Cloud Computing*, vol. 6, pp. 1067–1079, Oct 1 Oct.-Dec. 2018, doi: 10.1109/TCC.2016.2570750.

[59] M. Rost and S. Schmid, "Virtual network embedding approximations: Leveraging randomized rounding," *IEEE/ACM Transactions on Networking*, vol. 27, no. 5, pp. 2071–2084, Oct. 2019, doi: 10.1109/TNET.2019.2939950.

[60] 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*, vol. 15, no. 1, pp. 356–371, March 2018, doi: 10.1109/TNSM.2017.2778106.

[61] D. Dietrich, A. Rizk, and P. Papadimitriou, "Multi-provider virtual network embedding with limited information disclosure," *IEEE Transactions on Network and Service Management*, vol. 12, no. 2, pp. 188–201, 2015, doi: 10.1109/TNSM.2015.2417652.

[62] Y. Zhu and M. Ammar, "Algorithms for assigning substrate network resources to virtual network components," in *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pp. 1–12, 2006, doi: 10.1109/INFOCOM.2006.322.

[63] Long Gong, Yonggang Wen, Zuqing Zhu, and T. Lee, "Revenue-driven virtual network embedding based on global resource information," in *2013 IEEE Global Communications Conference (GLOBECOM)*, pp. 2294–2299, 2013, doi: 10.1109/GLOCOM.2013.6831416.

[64] L. Gong, Y. Wen, Z. Zhu, and T. Lee, "Toward profit-seeking virtual network embedding algorithm via global resource capacity," in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pp. 1–9, 2014, doi: 10.1109/INFOCOM.2014.6847918.

[65] M. R. Rahman, I. Aib, and R. Boutaba, "Survivable virtual network embedding," in *NETWORKING 2010* (M. Crovella, L. M. Feeney, D. Rubenstein, and S. V. Raghavan, eds.), pp. 40–52, Springer Berlin Heidelberg, 2010.

[66] M. Feng, J. Liao, J. Wang, S. Qing, and Q. Qi, "Topology-aware virtual network embedding based on multiple characteristics," in *2014 IEEE International Conference on Communications (ICC)*, Sydney, NSW, 2014, pp.2956-2962, doi: 10.1109/ICC.2014.6883774.

[67] P. Zhang, H. Yao, and Y. Liu, "Virtual network embedding based on the degree and clustering coefficient information," *IEEE Access*, vol. 4, pp. 8572–8580, 2016, doi: 10.1109/ACCESS.2016.2632421.

[68] H. Cao, L. Yang, and H. Zhu, "Embedding virtual networks using a novel node-ranking approach via exploiting topology attributes and global network resources," in *2017 9th International Conference on Wireless Communications and Signal Processing (WCSP)*, Nanjing, 2017, pp. 1-6, doi: 10.1109/WCSP.2017.8171175.

[69] H. Cao, L. Yang, and H. Zhu, "Novel node-ranking approach and multiple topology attributes-based embedding algorithm for single-domain virtual network embedding," *IEEE Internet of Things Journal*, vol. 5, pp. 108–120, Feb 2018, doi: 10.1109/JIOT.2017.2773489.

[70] H. Cao, Y. Zhu, L. Yang, and G. Zheng, "A efficient mapping algorithm with novel node-ranking approach for embedding virtual networks," *IEEE Access*, vol. 5, pp. 22054–22066, 2017, doi: 10.1109/ACCESS.2017.2761840.

[71] C. Zhao and B. Parhami, "Virtual network embedding through graph eigenspace alignment," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 632–646, June 2019, doi: 10.1109/TNSM.2019.2895354.

[72] J. Lischka and H. Karl, "A virtual network mapping algorithm based on subgraph isomorphism detection," in *Proceedings of the 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures*, VISA '09, New York, NY, USA, p. 81–88, Association for Computing Machinery, 2009.

[73] A. Xiao, Y. Wang, L. Meng, X. Qiu, and W. Li, "Topology-aware virtual network embedding to survive multiple node failures," in *2014 IEEE Global Communications Conference*, Austin, TX, 2014, pp. 1823-1828, doi: 10.1109/GLOCOM.2014.7037073.

[74] H. Yu, V. Anand, C. Qiao, and G. Sun, "Cost efficient design of survivable virtual infrastructure to recover from facility node failures," in *2011 IEEE International Conference on Communications (ICC)*, Kyoto, 2011, pp. 1-6, doi: 10.1109/icc.2011.5962604.

[75] C. Qiao, B. Guo, S. Huang, J. Wang, T. Wang, and W. Gu, "A novel two-step approach to surviving facility failures," in *Optical Fiber Communication Conference/National Fiber Optic Engineers Conference 2011*, p. OWAA3, Optical Society of America, 2011.

[76] Q. Hu, Y. Wang, and X. Cao, "Location-constrained survivable network virtualization," in *2012 35th IEEE Sarnoff Symposium*, Newark, NJ, 2012, pp. 1-5, doi: 10.1109/SARNOF.2012.6222734.

[77] H. Jiang, L. Gong, and Z. W. Zuqing, "Efficient joint approaches for location-constrained survivable virtual network embedding," in *2014 IEEE Global Communications Conference*, Austin, TX, 2014, pp. 1810-1815, doi: 10.1109/GLOCOM.2014.7037071.

[78] M. R. Rahman and R. Boutaba, "Svne: Survivable virtual network embedding algorithms for network virtualization," *IEEE Transactions on Network and Service Management*, vol. 10, pp. 105–118, June 2013, doi: 10.1109/TNSM.2013.013013.110202.

[79] M. M. Alam Khan, N. Shahriar, R. Ahmed, and R. Boutaba, "Multi-path link embedding for survivability in virtual networks," *IEEE Transactions on Network and Service Management*, vol. 13, no. 2, pp. 253–266, 2016, doi: 10.1109/TNSM.2016.2558598.

[80] M. M. Alam Khan, N. Shahriar, R. Ahmed, and R. Boutaba, "Simple: Survivability in multi-path link embedding," in *2015 11th International Conference on Network and Service Management (CNSM)*, Barcelona, 2015, pp. 210-218, doi: 10.1109/CNSM.2015.7367361.

[81] A. Todimala and B. Ramamurthy, "A scalable approach for survivable virtual topology routing in optical wdm networks," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 6, pp. 63–69, August 2007, doi: 10.1109/JSAC-OCN.2007.020605.

[82] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," *SIGCOMM Computer Communication Review*, vol. 41, p. 350–361, Aug. 2011, doi: 10.1145/2018436.2018477.

[83] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, Chen-Nee Chuah, and C. Diot, "Characterization of failures in an ip backbone," in *IEEE INFOCOM 2004*, Hong Kong, 2004, pp. 2307-2317 vol.4, doi: 10.1109/INFCOM.2004.1354653.

[84] N. Shahriar, R. Ahmed, S. R. Chowdhury, M. M. A. Khan, R. Boutaba, J. Mitra, and F. Zeng, "Virtual network embedding with guaranteed connectivity under multiple substrate link failures," *IEEE Transactions on Communications*, vol. 68, no. 2, pp. 1025-1043, Feb. 2020, doi: 10.1109/TCOMM.2019.2954410.

[85] L. Gong, H. Jiang, Y. Wang, and Z. Zhu, "Novel location-constrained virtual network embedding lc-vne algorithms towards integrated node and link mapping," *IEEE/ACM Transactions on Networking*, vol. 24, pp. 3648–3661, December 2016, doi: 10.1109/TNET.2016.2533625.

[86] T. Mano, T. Inoue, D. Ikarashi, K. Hamada, K. Mizutani, and O. Akashi, "Efficient virtual network optimization across multiple domains without revealing private information," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 477–488, Sept. 2016, doi: 10.1109/TNSM.2016.2582179.

[87] Y. Li and Y. Zhang, "Epvne: An efficient parallelizable virtual network embedding algorithm," *Hindawi Wireless Communications and Mobile Computing*, vol. 2019, p. 8416592, Nov 2019, doi: 10.1155/2019/8416592.

[88] Y. Liang and S. Zhang, "Embedding parallelizable virtual networks," *Elsevier Computer Communications*, vol. 102, pp. 47 – 57, Apr. 2017, doi: 10.1016/j.comcom.2017.01.007.

[89] C. K. Dehury and P. K. Sahoo, "Dyvine: Fitness-based dynamic virtual network embedding in cloud computing," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 5, pp. 1029–1045, May 2019, doi: 10.1109/JSAC.2019.2906744.

[90] X. Mi, X. Chang, J. Liu, L. Sun, and B. Xing, "Embedding virtual infrastructure based on genetic algorithm," in *2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies*, Beijing, Dec. 2012, pp. 239-244, doi: 10.1109/PDCAT.2012.71.

[91] X. Chang, X. Mi, and J. Muppala, "Performance evaluation of artificial intelligence algorithms for virtual network embedding," *Elsevier Engineering Applications of Artificial Intelligence*, vol. 26, no. 10, pp. 2540 – 2550, 2013, doi: 10.1016/j.engappai.2013.07.007.

[92] Z. Zhou, X. Chang, Y. Yang, and L. Li, "Resource-aware virtual network parallel embedding based on genetic algorithm," in *2016 17th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, Guangzhou, 2016, pp. 81-86, doi: 10.1109/PDCAT.2016.031.

[93] 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, p. 040308, Mar 2017, doi: 10.1007/s11432-016-9015-3.

[94] 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, Mar 2019, doi: 10.1007/s12083-017-0609-x.

[95] C. Huang, C. Shen, C. Huang, T. Chin, and S. Shen, "An efficient joint node and link mapping approach based on genetic algorithm for network virtualization," in *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*, pp. 1–5, Honolulu, HI, USA, 2019, pp. 1-5, doi: 10.1109/VTCFall.2019.8891475.

[96] C. Aguilar-Fuster, M. Zangiabady, J. Zapata-Lara, and J. Rubio-Loyola, "Online virtual network embedding based on virtual links' rate requirements," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1630–1644, Dec. 2018, doi: 10.1109/TNSM.2018.2873923.

[97] L. Boyang, W. Muqing, and Z. Haosen, "Virtual network embedding based on hybrid adaptive genetic algorithm," in *2019 IEEE 5th International Conference on Computer and Communications (ICCC)*, Chengdu, China, 2019, pp. 1197-1202, doi: 10.1109/ICCC47050.2019.9064173.

[98] P. Zhang, X. Pang, G. Kibalya, N. Kumar, S. He, and B. Zhao, "Gcmd: Genetic correlation multi-domain virtual network embedding algorithm," *IEEE Access*, vol. 9, pp. 67167–67175, 2021.

[99] I. Fajjari, N. A. Saadi, G. Pujolle, and H. Zimmermann, "Vne-ac: Virtual network embedding algorithm based on ant colony metaheuristic," in *2011 IEEE International Conference on Communications (ICC)*, Kyoto, Jun. 2011, pp. 1-6, doi: 10.1109/icc.2011.5963442.

[100] J. Wang, W. Chen, H. Cong, Z. Zhan, and J. Zhang, "An ant colony system based virtual network embedding algorithm," in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Banff, AB, 2017, pp. 1805-1810, doi: 10.1109/SMC.2017.8122878.

[101] Hong-Kun Zheng, J. Li, Y. Gong, W. Chen, Zhiwen Yu, Z. Zhan, and Ying Lin, "Link mapping-oriented ant colony system for virtual network embedding," in *2017 IEEE Congress on Evolutionary Computation (CEC)*, San Sebastian, June 2017, pp. 1223-1230, doi: 10.1109/CEC.2017.7969445.

[102] F. Zhu and H. Wang, "A modified aco algorithm for virtual network embedding based on graph decomposition," *Elsevier Computer Communications*, vol. 80, pp. 1 – 15, Apr. 2016, doi: 10.1016/j.comcom.2015.07.014.

[103] A. Song, W. Chen, T. Gu, H. Yuan, S. Kwong, and J. Zhang, "Distributed virtual network embedding system with historical archives and set-based particle swarm optimization," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pp. 1–16, Feb. 2021, doi: 10.1109/TSMC.2018.2884523.

[104] A. Song, W. N. Chen, and X. M. Hu, "One-stage and dual-heuristic particle swarm optimization for virtual network embedding," in *2020 IEEE Congress on Evolutionary Computation (CEC)*, Glasgow, United Kingdom, 2020, pp. 1-7, doi: 10.1109/CEC48606.2020.9185524.

[105] X. Liu, Z. Zhang, X. Li, and S. Su, "Optimal virtual network embedding based on artificial bee colony," *EURASIP Journal on Wireless Communications and Networking*, vol. 2016, p. 273, Nov 2016, doi: 10.1186/s13638-016-0766-2.

[106] I. Pathak, A. Tripathi, and D. P. Vidyarthi, "A model for virtual network embedding using artificial bee colony," *Wiley International Journal of Communication Systems*, vol. 31, no. 10, p. e3573, 2018, doi: IJCS-17-0210.R2.

[107] L. Wang, H. Qu, J. Zhao, and Y. Guo, "Virtual network embedding with discrete particle swarm optimisation," *IET Electronics Letters*, vol. 50, no. 4, pp. 285–286, 2014, doi: 10.1049/el.2013.3202.

[108] Y. Yuan, C. Wang, C. Wang, S. Zhu, and S. Zhao, "Discrete particle swarm optimization algorithm for virtual network reconfiguration," in *Springer Advances in Swarm Intelligence*, (Berlin, Heidelberg), pp. 250–257, Berlin Heidelberg, 2013.

[109] Y. Yuan, C.-R. Wang, C. Wan, C. Wang, and X. Song, "Repeatable optimization algorithm based discrete pso for virtual network embedding," in *Springer Advances in Neural Networks – ISNN 2013*, (Berlin, Heidelberg), pp. 334–342, Berlin Heidelberg, 2013.

[110] P. Zhang, H. Yao, C. Fang, and Y. Liu, "Multi-objective enhanced particle swarm optimization in virtual network embedding," *EURASIP Journal on Wireless Communications and Networking*, vol. 2016, p. 167, Jul 2016, doi: 10.1186/s13638-016-0669-2.

[111] K. Guo, Y. Wang, X. Qiu, W. Li, and A. Xiao, "Particle swarm optimization based multi-domain virtual network embedding," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, Ottawa, ON, 2015, pp. 798-801, doi: 10.1109/INM.2015.7140379.

[112] Y. Ni, G. Huang, S. Wu, C. Li, P. Zhang, and H. Yao, "A pso based multi-domain virtual network embedding approach," *China Communications*, vol. 16, no. 4, pp. 105–119, Apr. 2019.

[113] J. Rubio-Loyola, C. Aguilar-Fuster, G. Toscano-Pulido, R. Mijumbi, and J. Serrat-Fernández, "Enhancing metaheuristic-based online embedding in network virtualization environments," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 200–216, March 2018, doi: 10.1109/TNSM.2017.2742666.

[114] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, p. 16, Jun 2018, doi: 10.1186/s13174-018-0087-2.

[115] A. Blenk, P. Kalmbach, J. Zerwas, M. Jarschel, S. Schmid, and W. Kellerer, "Neurovine: A neural preprocessor for your virtual network embedding algorithm," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, Honolulu, HI, 2018, pp. 405-413, doi: 10.1109/INFOCOM.2018.8486263.

[116] S. Haeri and L. Trajković, "Virtual network embedding via monte carlo tree search," *IEEE Transactions on Cybernetics*, vol. 48, pp. 510–521, Feb 2018, doi: 10.1109/TCYB.2016.2645123.

[117] H. Yao, B. Zhang, P. Zhang, S. Wu, C. Jiang, and S. Guo, "Rdam: A reinforcement learning based dynamic attribute matrix representation for virtual network embedding," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2018, doi: 10.1109/TETC.2018.2871549.

[118] R. Buyya, C. Vecchiola, and S. T. Selvi, *Mastering Cloud Computing: Foundations and Applications Programming*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1st ed., 2013.

[119] A. S. Tanenbaum and M. van Steen, *Distributed Systems: Principles and Paradigms*. Upper Saddle River, NJ: Pearson Prentice Hall, 2 ed., 2007.

[120] K. Deb, *Multi-objective Optimisation Using Evolutionary Algorithms: An Introduction*. London: Springer London, 2011.

[121] M. Sipper, R. S. Olson, and J. H. Moore, "Evolutionary computation: the next major transition of artificial intelligence?," *BioData Mining*, vol. 10, p. 26, Jul 2017, doi: 10.1186/s13040-017-0147-3.

[122] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Springer Publishing Company, Incorporated, 2nd ed., 2015, doi: 10.1007/978-3-662-44874-8.

[123] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, Mar. 1998, ISBN: 9780262631853.

[124] J. Shapiro, *Genetic Algorithms in Machine Learning*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, ISBN: 978-3-540-44673-6.

[125] X.-S. Yang, "Chapter 5 - genetic algorithms," in *Nature-Inspired Optimization Algorithms* (X.-S. Yang, ed.), pp. 77 – 87, Oxford: Elsevier, 2014, doi: 10.1016/B978-0-12-416743-8.00005-1.

[126] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," Sept. 2017, arXiv:1703.03864.

[127] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," Apr. 2018, arXiv:1712.06567.

[128] K. Nguyen and C. Huang, "Distributed parallel genetic algorithm for online virtual network embedding," *Wiley International Journal of Communication Systems*, 23 Dec 2020, pp. e4691, doi: 10.1002/dac.4691.

[129] K. Nguyen, Q. Lu, and C. Huang, "Rethinking virtual link mapping in network virtualization," in *2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall)*, Victoria, Canada, Nov. 2020.

[130] F. Hansen and G. K. Pedersen, "Jensen's Operator Inequality," *Bulletin of the London Mathematical Society*, vol. 35, issue. 4, no. 4, pp. 553–564, Jul. 2003, doi: 10.1112/S0024609303002200.

[131] C. Molnar and J. Gair, "Concepts of biology - 1st canadian edition," *BCcampus, Victoria, B.C.*, May 2015, ISBN: 978-1-989623-99-2.

[132] K. Nguyen, Q. Lu, and C. Huang, "Joint node-link embedding algorithm based on genetic algorithm in virtualization environment," in *2021 IEEE 94th Vehicular Technology Conference (VTC2021-Fall)*, Virtual conference, Sep. 2021 (accepted for publication).

[133] K. Nguyen and C. Huang, "Towards joint node and link mapping algorithms for embedding virtual networks: A conciliation strategy," *IEEE Transactions on Network and Service Management*, 2021 (submitted).

[134] K. Cao, Y. Liu, G. Meng, and Q. Sun, "An overview on edge computing research," *IEEE Access*, vol. 8, pp. 85714–85728, 2020, doi: 10.1109/ACCESS.2020.2991734.

[135] X. Huang, R. Yu, J. Liu, and L. Shu, "Parked vehicle edge computing: Exploiting opportunistic resources for distributed mobile applications," *IEEE Access*, vol. 6, pp. 66649–66663, 2018, doi: 10.1109/ACCESS.2018.2879578.

[136] F. H. Rahman, A. Yura Muhammad Iqbal, S. H. S. Newaz, A. Thien Wan, and M. S. Ahsan, "Street parked vehicles based vehicular fog computing: Tcp throughput evaluation and future research direction," in

2019 21st International Conference on Advanced Communication Technology (ICACT), PyeongChang Kwangwoon Do, Korea (South), 2019, pp. 26-31, doi: 10.23919/ICACT.2019.8701912.

[137] D. Han, W. Chen, and Y. Fang, "A dynamic pricing strategy for vehicle assisted mobile edge computing systems," *IEEE Wireless Communications Letters*, vol. 8, no. 2, pp. 420–423, April 2019, doi: 10.1109/LWC.2018.2874635.

[138] O. Fadahunsi and M. Maheswaran, "Locality sensitive request distribution for fog and cloud servers," *Springer Service Oriented Computing and Applications*, vol. 13, pp. 127–140, Jun 2019, doi: 10.1007/s11761-019-00260-2.

[139] "What is kubernetes?." https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/. Accessed: 2020-05-28.

[140] S. Arif, S. Olariu, J. Wang, G. Yan, W. Yang, and I. Khalil, "Datacenter at the airport: Reasoning about time-dependent parking lot occupancy," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 11, pp. 2067–2080, Nov. 2012, doi: 10.1109/TPDS.2012.47.

[141] W. He, G. Yan, and L. D. Xu, "Developing vehicular data cloud services in the iot environment," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1587–1595, May 2014, doi: 10.1109/TII.2014.2299233.

[142] F. Dressler, P. Handle, and C. Sommer, "Towards a vehicular cloud - using parked vehicles as a temporary network and storage infrastructure," in *Proceedings of the 2014 ACM International Workshop on Wireless and Mobile Technologies for Smart Cities*, WiMobCity '14, (New York, NY, USA), p. 11–18, Association for Computing Machinery, 2014.

[143] E. Al-Rashed, M. Al-Rousan, and N. Al-Ibrahim, "Performance evaluation of wide-spread assignment schemes in a vehicular cloud," *Elsevier Vehicular Communications*, vol. 9, pp. 144 – 153, Jul. 2017, doi: 10.1016/j.vehcom.2017.05.005.

[144] T. Kim, H. Min, and J. Jung, "Vehicular datacenter modeling for cloud computing: Considering capacity and leave rate of vehicles," *Elsevier Future Generation Computer Systems*, vol. 88, pp. 363 – 372, 2018, doi: 10.1016/j.future.2018.05.052.

[145] C. Li, S. Wang, X. Huang, X. Li, R. Yu, and F. Zhao, "Parked vehicular computing for energy-efficient internet of vehicles: A contract theoretic approach," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6079–6088, Aug. 2019, doi: 10.1109/JIOT.2018.2869892.

[146] X. Hou, Y. Li, M. Chen, D. Wu, D. Jin, and S. Chen, "Vehicular fog computing: A viewpoint of vehicles as the infrastructures," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 6, pp. 3860–3873, June 2016, doi: 10.1109/TVT.2016.2532863.

[147] X. Wang, Z. Ning, and L. Wang, "Offloading in internet of vehicles: A fog-enabled real-time traffic management system," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4568–4578, Oct. 2018, doi: 10.1109/TII.2018.2816590.

[148] X. Huang, P. Li, and R. Yu, "Social welfare maximization in container-based task scheduling for parked vehicle edge computing," *IEEE Communications Letters*, vol. 23, no. 8, pp. 1347–1351, Aug. 2019, doi: 10.1109/LCOMM.2019.2920832.

[149] Y. Cao, Y. Teng, F. R. Yu, V. C. M. Leung, Z. Song, and M. Song, "Delay sensitive large-scale parked vehicular computing via software defined blockchain," in *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, May Seoul, Korea (South), 2020, pp. 1-6, doi: 10.1109/WCNC45663.2020.9120494.

[150] S. Raza, W. Liu, M. Ahmed, M. R. Anwar, M. A. Mirza, Q. Sun, and S. Wang, "An efficient task offloading scheme in vehicular edge computing," *Journal of Cloud Computing*, vol. 9, p. 28, Jun 2020.

[151] K. Nguyen, S. Drew, C. Huang, and J. Zhou, "Collaborative container-based parked vehicle edge computing framework for online task offloading," in *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*, pp. 1–6, 2020.

[152] H. Zhu and C. Huang, "VNF-B&B: Enabling edge-based NFV with CPE resource sharing," in *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pp. 1–5, 2017.

[153] K. Nguyen, S. Drew, C. Huang, and J. Zhou, "Edgepv: Collaborative edge computing framework for task offloading," in *ICC 2021 - 2021 IEEE International Conference on Communications (ICC)*, pp. 1–6, 2021.

[154] K. Nguyen, S. Drew, C. Huang, and J. Zhou, "Parked vehicles task offloading in edge computing," *IEEE Access*, pp. 01–13, 2021.

[155] K. Nguyen, Q. Lu, and C. Huang, "Efficient virtual network embedding with node ranking and intelligent link mapping," in *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*, Nov. 2020.