Development of Dynamic Resource Management Mechanisms for Virtualized Network Environments

by

Laaziz LAHLOU

MANUSCRIPT-BASED THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
Ph.D.

MONTREAL, APRIL 15, 2021

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Mrs. Nadjia Kara, Thesis supervisor
Department of Software Engineering and IT, École de Technologie Superieure

Mr. Chakib Tadj, President of the board of examiners
Department of Electrical Engineering, École de Technologie Superieure

Mr. Aris Leivadeas, Member of the jury
Department of Software Engineering and IT, École de Technologie Superieure

Mr. Chamseddine Talhi, Member of the jury
Department of Software Engineering and IT, École de Technologie Superieure

Mr. Wahab Hamou-Lhadj, External examiner
Department of Electrical and Computer Engineering at Gina Cody School of Engineering and Computer Science, Concordia University

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE PUBLIC

ON APRIL 12, 2021

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

FOREWORD

The present doctoral thesis is structured in the form of a compilation of articles. Throughout this research work, five journal articles were written and submitted. They are integrated with high fidelity to ensure compliance with the proposed and published articles' structure and shape. Still, only peripheral modifications (e.g., figures framing, repositioning, and rescaling) were made under Ecole de Technologie Superieure's thesis template. While each article addresses different aspects of the research problems, the total contributions are closely interrelated. In addition to the incorporated journal articles, a specific section is devoted to the literature with the most recent and closest publications to the research problems covered within this doctoral thesis. Henceforth, each article comprises a specialized literature review at the periphery of the problem formulation. Notwithstanding, the set of papers is embroidered with an introduction, a conclusion, and many future research avenues to extend the current contributions.

ACKNOWLEDGEMENTS

I want to express my gratitude to my supervisor, Professor Nadjia Kara, for all her support during these four years of doctoral studies. Dr. Nadjia Kara has been amazingly patient to allow me to grow, learn and mature as a researcher. Her invaluable help made this work possible. I am not sure this could be possible without her.

I want to thank our industrial partners who have been with us since the beginning of this Ph.D. journey for their feedback and comments that helped shape this dissertation.

I want to thank Professor Raouf Boutaba, Professor Ahmed Mehaoua, Professor Hassine Moungla for their trust, patience, and the opportunities they provided me through my graduate career at Paris Descartes University. I have been more than fortunate to have learned from them. I would also like to thank my professors at Strasbourg University, Bejaia University, and all the educators/teachers/mentors I have had throughout my academic career since age 6!

I extend my thanks to my examining committee for reviewing my thesis and participating in my defense. I extend my sincere thanks to Mr. Chakib Tadj, Mr. Wahab Hamou-Lhadj, Mr. Aris Leivadeas, and Mr. Chamseddine Talhi for agreeing to evaluate this work as well as for their advice and guidance.

My deepest appreciation goes to my parents, Abdel-Hakem and Houria, for their unconditional love, for all the sacrifices throughout my life that they have been able to make to educate us and raise us to where it was not possible for them because of me, my brother and my sisters. My father is the reason if I am in computer science. My parents are at the origin of my perseverance since my childhood towards studying. I thank my parents for their patience and resilience due to the pain I have inflicted on them because of my distance from them in search of knowledge.

I thank my wife Lynda for her love, support, understanding, and patience throughout my Ph.D. journey. She has always been telling me: "Your work behind the black screen, when coding, does not make me understand anything because I am not a computer science specialist, but your hands on the keyboard and your intelligence will make your work relevant and of great

success with your love for what you do in computer science. Be courageous, persevere, and run behind knowledge because behind this black screen, there will always be a white screen that will illuminate your path to success". Words will not be enough to thank her.

I thank all my friends and colleagues in the vantrix lab with whom I started this P.hD journey in January 2017, with whom I have spent joyful moments and nice discussions beyond research.

I thank my cousins, my uncles, my aunts, grandfather, grandmother, brother, sisters, and people who are dear to me. Special thanks go to uncle Idir and his wife for the joyful moments we spent together throughout my Ph.D. journey. I want to thank my uncle Houcine and his family.

I dedicate this humble thesis to the soul of my grandmother, my uncle, Si Tahar, Si Mohand, my parents-in-law, my brothers-in-law, my sister-in-law, my friends in Algeria, my friends in France, my friend in Canada and all the people whose life, as difficult as it is, has prevented them from studying and having access to education.

Développement de mécanismes de gestion dynamique des ressources pour les environnements de virtualisation de réseau

Laaziz LAHLOU

RÉSUMÉ

La croissance sans précédent et extrême de l'informatique dans les nuages, la virtualisation des fonctions réseau et les paradigmes de gestion de réseau définis par logiciel ont conduit et guidé de nombreuses développements technologiques. Ces avancées ont permis de revoir la conception, le déploiement et la gestion des services réseau de manière significative et -quelque peu- radicale. Malgré les promesses en termes de réduction des coûts d'exploitation, d'automatisation des opérations et d'augmentation des revenus, plusieurs défis restent à relever pour pouvoir assurer une transition vers l'automatisation complète de la gestion du réseau et des services.

Parmi ces enjeux, le placement et le chaînage des fonctions réseau virtualisées ainsi que l'adaptation dynamique de celles-ci restent parmi les plus étudiés et traités dans la literature. En pratique, ces deux problèmes sont considérés comme très prometteurs pour réussir cette transition. En même temps, ils sont interconnectés et nécessitent des solutions efficaces qui répondent à la fois aux demandes des applications et des services auxquels ils sont destinés, lorsqu'ils sont déployés et reconfigurés pour s'adapter à certaines évolutions à moindre coût.

Dans cette thèse, les deux problèmes sont étudiés et différentes techniques ont été proposées pour les résoudre. Nous avons envisagé différents scénarios pour chaque problème et évalué à l'aide de plusieurs mesures de performance. En outre, cette dissertation contient quatre contributions: Premièrement, FASTCALE, un algorithme génétique culturel évolutif est proposé pour le placement et le chaînage de services de réseaux virtuels complexes; Deuxièmement, ARTIMIS une suite de techniques d'optimisation pour la sélection des VNF, compte tenu de leurs différentes performances et configurations, en ce qui concerne les services sensibles aux délais et une méta-heuristique basée sur le principe de réactions chimiques pour leur placement et chaînage. Troisièmement, VALKYRIE un ensemble de techniques de partionnement permettant le déploiement de chaînes de fonctions virtualisées à travers des grappes de noeuds à la demande afin de pouvoir assurer la réduction de l'espace de recherche vers des solutions réalisables garanties; Quatrièmement, DAVINCI une approche d'adaptation dynamique qui intègre des mécanismes d'élasticité comme un ensemble de décisions pour adapter rapidement les chaînes de fonctions de service à moindre coût. Les résultats de l'évaluation sont présentés pour montrer l'efficacité des modèles et algorithmes proposés.

Mots-clés: virtualisation de fonctions réseaux, allocation de ressources, chaînes de service dynamiques, placement de fonction réseaux, algorithmes exacte et stochastique

Development of Dynamic Resource Management Mechanisms for Virtualized Network Environments

Laaziz LAHLOU

ABSTRACT

The unprecedented and extreme cloud computing growth, virtualization of network functions, and software-defined networking paradigms have driven and guided many technological developments. These advances made it possible to significantly and somewhat radically review the design, deployment, and management of network services.

Despite the promises in terms of reduced operating costs, automated operations, and increased revenue, several challenges remain to overcome to be able to make the transition to full automation of network and service management. Among the challenges, the placement and chaining of virtualized network functions and the dynamic adaptation of these remain among the most studied. In practice, these two problems are considered very promising for making this transition successful. Simultaneously, they are interconnected and require effective and practical solutions that meet the demands of the applications and services they are intended for when deployed and reconfigured to accommodate specific changes at a lower cost.

In this thesis, the two problems are studied, and different techniques have been proposed to solve them. We have considered different scenarios for each research problem and evaluated using several performance metrics. Furthermore, it contains four contributions: First, FASTCALE, a scalable cultural genetic algorithm is proposed for the placement and chaining of complex virtual network services; Second, ARTIMIS a suite of optimization techniques for the selection of VNFs, given their different performances and configurations, for delay-sensitive services and a chemical reaction based meta-heuristic for their placement and chaining. Third, VALKYRIE a set of clustering techniques enabling the deployment of service function chains across on-demand clusters and the reduction of the search space toward guaranteed feasible solutions; Forth, DAVINCI a dynamic adaptation approach that incorporates elasticity mechanisms as a set of decisions to adapt the service function chains with least cost quickly. The evaluation results are presented to show the effectiveness of the proposed models and algorithms.

Keywords: network function virtualization, resource allocation, dynamic service function chaining, VNF placement, exact and stochastic algorithms

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

LIST OF ALGORITHMS

Page

LIST OF ABBREVIATIONS

ETS	École de Technologie Supérieure
NFV	Network Function Virtualization
SDN	Software Defined Networking
SFC	Service Function Chain
VM	Virtual Machine
CRO	Chemical Reaction Optimization
GT	Game Theory
E-GT	Enhanced Game Theory
VNF	Virtual Network Function
CPU	Central Processing Unit
RAM	Random Access Memory
ETSI	European Telecommunications Standards Institute
SLO	Service Level Objective
NP	Non-Polynomial
CGA	Cultural Genetic Algorithm
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
ILP	Integer Linear Programming
MILP	Mixed Integer Linear Programming
IPTV	Internet Protocol Television

IEGWO	Integer Encoding Grey Wolf Optimizer
DAFT	DynAmic vnF placemenT
RA-RA	Resource Aware Routing Algorithm
BILP/BIP	Binary Integer Linear Programming
SDK	Software Development Kit
QoS	Quality of Service
IMS	IP Multimedia Subsystem
OPEX	OPerational EXpenditure
CAPEX	CApital EXpenditure
NSGA-II	Non Dominated Sorting Genetic Algorithm-II
MA	Markov Approximation
SLA	Service Level Agreement
CDN	Content Delivery Network
P-CSCF	Proxy Call Session Control Function
I-CSCF	Interrogating Call Session Control Function
S-CSCD	Serving Call Session Control Function
HSS	Home subscriber Server
VNS	Virtualized Network Service
GenoFix	Genotype Fixing
MANO	MANagement and Orchestration

NFVO	NFV Orchestrator
VNFM	Virtual Network Function Manager
NSERC	Natural Sciences and Engineering Research Counci of Canada
KPI	Key Performance Indicator
SFC-MAP	SFC embedding Algorithm
VNF-DRA	VNF dynamic release algorithm
FIFS	First-In First-Served
SR	Stable Roommate
BFS	Breadth First Search
LM	Lagrange Multiplier
OPAC	Optimal Placement Algorithm for virtual CDN
HPAC	Heuristic Placement Algorithm for virtual CDN

INTRODUCTION

0.1 Context and motivation

Casado, Freedman, Pettit, Luo, McKeown & Shenker (2007) paved the way with Open source technologies to a new trend and paradigm shift and opened doors to many innovations within networking and systems. His seminal work leads to rethinking software and hardware designs towards delivering novel applications and network services, followed by a vast community working together towards the next generation architectures of the future network. In the last decade, the most prominent trends and paradigm shifts driving the academic and industrial areas are Software-Defined Networking (SDN) and Network Functions Virtualization (NFV) Bonfim, Dias & Fernandes (2019a), respectively. While Software Defined Networking redefined the way, physical middleboxes should operate and behave by separating the coupled and locked control plane and data plane. Network Functions Virtualization, in turn, proposes to use virtualization technology for running the network functions (e.g., fire-walling, natting, deep packet inspections, intrusion detection, and prevention systems) inside virtual machines or containers that run on top of physical servers or commodity hardware machines. Yet, it enables both the decoupling from legacy (dedicated) hardware and ease of management of the network functions as mentioned above.

Both NFV and SDN brought out new abstraction levels to improve and simplify the management and the integration of networks with significant impacts on the business revenues and how network architects redefined the networking topology on top of which added value services are delivered to customers/end users. Most network functions need a set of resources (e.g., CPU cores, memory, and storage) to be run virtually to process the incoming traffic that traverses them, and virtual machines or physical servers provide them. Previously, network functions that have formed most of the network services were proprietary and legacy hardware. They were hand-wired, and their topology remained static, making the topology evolution even tricky,

and adding new legacy network functions becomes a tedious task. In addition to that, skillful persons are needed to manage and maintain the whole network infrastructure. Thanks to NFV, all of these tasks can be done automatically and bring the ease of management and deployment of network services that reduce both times to market and operational expenditure on a long-term basis as expected. Telecoms and service providers chose ETSI to lead NFV specification and standardization Mijumbi, Serrat, Gorricho, Bouten, Turck & Boutaba (2016).

0.1.1 Problem statement

One of the most challenging problem in NFV is the resource allocation problem Weerasiri, Barukh, Benatallah, Sheng & Ranjan (2017); Gil Herrera & Botero (2016). The problem relates to the dynamic placement and adaptation of Virtual Network Functions and their chaining in a substrate physical network with the requested resources, i.e., processing power, memory, and bandwidth. This NFV concept is appealing, but several challenges remain and have to be overcome to fully benefit from such a new paradigm.

From a client's perspective, the placement and chaining of the VNFs consist in finding the appropriate substrate nodes with sufficient compute resources (e.g., CPU, RAM, and Disk Storage) the paths interconnecting them with the required bandwidth and end-to-end-delay demands. However, from a network's operator perspective, it consists of finding a feasible deployment that satisfies the client's request requirements and better optimizes its objectives, such as optimizing the resource consumption of the infrastructure and avoiding violating the Service Level Objectives (SLO). Each network operator may want to optimize either one or several goals, depending on its policy and strategies. Therefore, the same approach could be tuned to perform the the placement and chaining but with different optimization goals leading to another placement and chaining results.

The nature of the underlying infrastructure is evolving in terms of service availability and resources to host the virtualized functions. Moreover, such services' continually changing requirements mean that their deployment requires dynamic mechanisms to adapt to any changes that may occur over time. Hence, there is a need for solutions that enable the adaptation of the service function chains with little perturbations (e.g., VNF and link relocations) and least violation costs (e.g., migration downtime, end-to-end delay) when involving elasticity mechanisms (e.g., vertical and horizontal scaling) and migration procedure. Figure 0.1 depicts the effects of such adaptation mechanisms on an already deployed web service that provides access to online shopping which is composed of a NAT, a firewall, a traffic monitor, and a WAN optimizer.

The figure on the left shows its current deployment in a given network. Let us assume that the number of user requests increases and the traffic monitor becomes overloaded. Scalability mechanisms such as horizontal scaling (depicted by the figure in the middle) could help by adding a new instance to cope with this increasing number of requests. This operation involves finding a feasible assignment that corresponds to a server, with sufficient resources, and a path connecting it to the other VNFs with enough bandwidth capacity while load-balancing the traffic among the allocated paths.

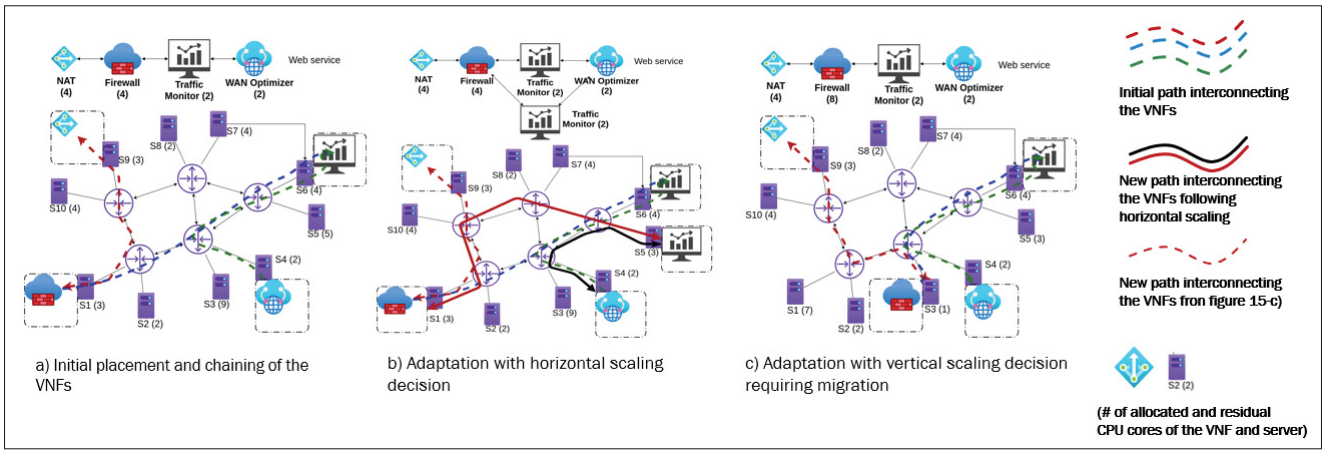


Figure 0.1 Exemple of an adaptation of a virtual network service (e.g., web service)

Similarly, if it is the firewall that gets overloaded, vertical scaling (depicted by the figure on the right) could help by adding more resources. However, in this example, the current server can not afford the operation. Hence, the VNF is migrated to another location. Therefore, the paths that originate from the migrated VNF would have to be adjusted.

Several approaches have been proposed in the literature for the placement and adaptation of service chains since the inception of NFV. Still, time-efficient, and cost-effective solutions are necessary to help network operators ease the orchestration of virtualized network functions composing the services regardless of their sizes and topology types.

0.1.2 Research objectives

The objectives that we set to achieve within the framework of this thesis stem from two open problems in the context of virtualized environments. While the former subject concerns the dynamic placement and chaining of VNFs of different sizes to compose various SFC topologies, the latter concerns their dynamic adaptation, coupled with elasticity mechanisms and migration procedure.

From an optimization perspective, the problems mentioned above are hard to solve and often reduced to well-known optimization problems (e.g., the Knapsack problem) Yang, Li, Trajanovski, Yahyapour & Fu (2021), which are considered to be NP-Hard, to obtain the optimal solution. Furthermore, a solution with an optimality guarantee is only possible for small instances of the problem because the execution time magnifies exponentially with the sizes of the infrastructure and the service function chain. Nevertheless, from a practical viewpoint, a solution with the optimality certificate is not sufficient to be seen as a deployable solution considering that today's infrastructures are massive. Therefore, techniques that trade optimality with convergence time towards practical and deployable solutions are more privileged by network operators to quickly

serve their clients' requests while satisfying their service level objectives as specified by the service level agreement contract.

We have addressed the two problems separately, but the resulting models are interrelated. We have extended the mathematical model we developed for the dynamic placement and chaining of the VNFs to account for the elasticity mechanisms and migration procedure for their dynamic adaptations. Moreover, we have designed low-algorithmic complexity (time-complexity) and sub-optimal techniques with scalability aptitude.

Figure 0.2 depicts the notion of the form we introduce within this thesis along with the quality of existing solutions, including our contributions, that can be found in the literature. Throughout this thesis, we concluded that given any approach for the placement and chaining of the VNFs, the returned feasible solution exhibits the same patterns, that is to say, either the VNFs are distributed or consolidated over the compute nodes as sketched by figure 0.2, be it optimal or sub-optimal, with or without theoretical guarantees, whatever the objective functions and constraints defined by the optimization model.

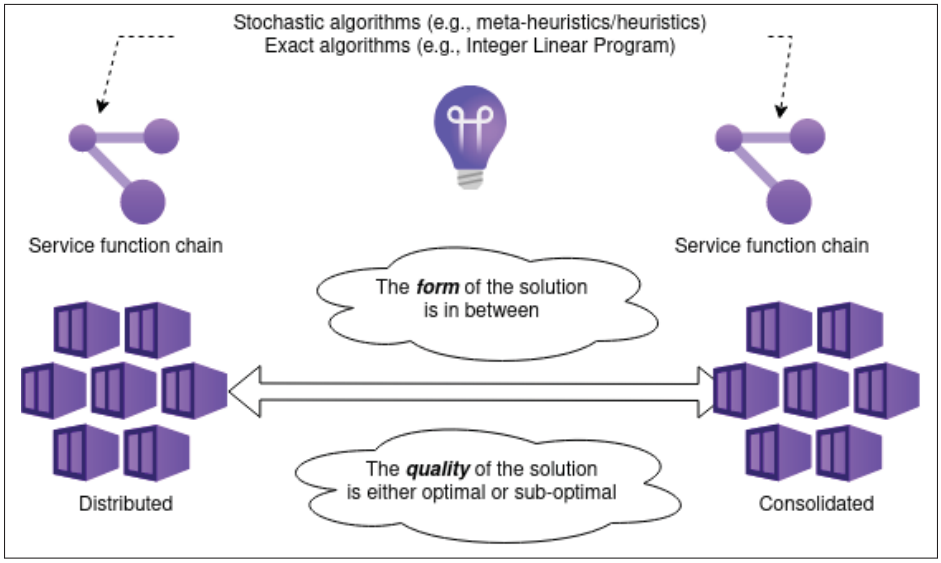


Figure 0.2 The form and quality of resulting solutions for the orchestration problem by any approach

Hereunder, we state the different objectives, and research goals concerning each problem addressed throughout this thesis where the solutions must be, in addition to satisfying the requirements of services and the optimization of infrastructure's resources:

1) **Time-efficient** (e.g., in terms of algorithmic complexity) to scale with the sizes of the service function chain and the underlying infrastructure at the same time;
2) **Generic** (e.g., applicable to different services with various topology being either linear or non-linear);
3) **Dynamic** to meet the evolving state of the services and infrastructure in terms of resource consumption and availability;
4) **Adaptive** to account for elasticity mechanisms (e.g., vertical and horizontal scaling) and migration procedure;
5) **Cost-effective** when orchestrating the services (e.g., deploying and adapting the services) from a network operator's perspective with the least costs (e.g., energy consumption, SLO violation).

It is worth mentioning that throughout this thesis, we use "placement and chaining of the VNFs" and "service function chain," "substrate physical network," and "infrastructure" indifferently.

0.2 Contributions

The present thesis's main objective is to propose novel and advanced algorithms for the dynamic orchestration of virtualized service function chains in terms of placement and adaptation, of any size and types of topologies, in heterogeneous and evolving cloud network environments. The proposed solutions must comply with the requirements of both the virtualized service function chains, resource demands and quality of service, and the costs of their deployments in the cloud settings in which they will be provisioned.

The proposed contributions were motivated and guided by the limitations of the existing approaches following our rigorous analysis of the state-of-the-art. All the existing solutions for the dynamic placement and adaptation of virtualized service function chains are limited to linear topologies, and their sizes do not exceed four virtualized network functions. Hence, the non-scalability aptitude of the current techniques and the algorithmic complexity increases with the infrastructures' dimensions and the service function chain requests. Furthermore, in a dynamic environment where resource usage evolves, the stringent requirements of the SLO must be fulfilled for the service function chains in addition to the required resources (e.g., CPU, Memory, Storage capacity, Bandwidth, and End-to-End delay). To cope with these technical issues, network operators enjoy the benefits of using elasticity mechanisms (e.g., vertical and horizontal scaling) and migration procedures to dynamically adapt the service function chains to meet their requirements while reducing the penalty costs continually. However, existing technical solutions do not evaluate such mechanisms' overall impact on the service function chains or the resulting penalty costs (e.g., migration downtime and SLO violation). Yet, all the surveyed studies consider migration, vertical and horizontal scaling of VNFs in isolation. The main contributions of the present thesis are outlined hereunder.

In the first paper, we proposed FASTCALE, a scalable multi-objective constrained decision-making optimization algorithm capable of overcoming the existing technical issues related to the placement and chaining of the virtual network functions. The proposed approach is based on an advanced metaheuristic (Cultural Algorithm) designed to work with different topologies of services and not limited to four VNFs thanks to a tailored genotype encoding scheme. Moreover, it enables expressing the existing affinity and anti-affinity constraints between the virtual network functions.

In the second paper, we proposed a variant of the FASTSCALE model targeted toward delay-sensitive service function chains titled ARTIMIS, a two-stage approach. In the first stage,

ARTIMIS selects the appropriate VNFs, from a pool of VNFs owned by a network operator, for the desired performance levels before their placements. Secondly, ARTIMIS uses a chemical reaction optimization method for the placement and chaining of the previously selected VNFs.

In the third paper, as an attempt to decrease the convergence time of the placement and chaining algorithms, we proposed VALKYRIE, a suite of unsupervised learning techniques, to help partition cloud infrastructures into on-demand clusters with reduced sizes while being adapted to their specific demands and requirements in terms of resources and networking capabilities. In this way, we will be able to reduce the execution time of our approach to placing and chaining VNFs and guarantee that a solution will be found which meets the requirements of the service chain's requests. This is not the case if we do not split the network according to the service chains' needs. Indeed, since an evolutionary algorithm is stochastic by nature, it may not find a feasible solution on the first attempt because the size of the network is huge, and the possibilities for placing VNFs are immense. Therefore, several attempts might be necessary. However, this situation can be avoided with on-demand partitioning techniques that reduce the search space significantly. The proposed solutions leveraged Graph and Game Theory formalisms and compared them against well-known unsupervised learning techniques such as K-Means and DBSCAN.

In the fourth paper, we proposed DAVINCI, a decision-making tool with different adaptation policies that allow the network operators to adapt the service function chains to minimize network changes (e.g., reallocation of paths and VNFs) while keeping the penalty costs at their lowest level. DAVINCI enables expressing the adaptations (e.g., migration, vertical and horizontal scaling) as a set of decisions.

A common denominator of existing solutions is their non-scalability aptitude and higher algorithmic complexity. In the fifth paper, we set to tackle the limitations mentioned earlier by formulating the placement and chaining of VNFs as a graph matching problem. This

formulation's intuition is to leverage the graph-aided representation of the service function chains to develop time-efficient and scalable techniques that would identify the existing and already orchestrated service function chains with graph representations that seem to be similar. Further to this, the ability to reuse the previous experience and computations of existing decision-making techniques (e.g., genetic algorithms, particle swarm optimization, heuristics/metaheuristics). Henceforth, the proposal of an edge-preserving graph matching and a matrix-based similarity computation score algorithm pinpoints the most similar service function chains. Mention should be made of the non-stochastic nature of the proposed algorithms.

Last but not least, we are investigating, with the help of a Master student, the implementation of an advanced data-structure that enables us to design a time-efficient and scalable algorithm, which does not require performing path computations to chain a pair of VNFs similar to our previous contributions and most of the existing solutions in the state-of-the-art Yang *et al.* (2021). Indeed, chaining the VNFs is an essential operation of a typical orchestration technique, which often makes use of the shortest path algorithm where the best implementation runs in $\mathcal{O}(E \times Log(V) + V)$ using priority queues for Dijkstra. Yet, this operation is not free of charge as it impacts the overall time-efficiency of the solution. However, our proposed approach does not require invoking any existing shortest path algorithms. Henceforth, the low algorithmic complexity aptitude of our novel method. It is worth mentioning that these last two contributions are not included in the current manuscript because these are conference papers.

0.3 Methodology

For all the technical contributions made in the present thesis, we followed a similar methodology to each solution, which consisted of succinctly conducting a literature review, the problem formulation followed by its modeling by mathematical formalisms before being solved using various techniques (e.g., exact methods (e.g., Integer Linear Programming), stochastic algorithms (e.g., heuristics/metaheuristics)). The validation of the proposed solutions was carried out using

different scenarios and compared with the closest state-of-the-art techniques. The evaluation was conducted following the same settings similar to most state-of-the-art methods and the inputs provided by our industrial partners with synthetic tools (e.g., NetworkX).

Figure 0.3 depicts the chosen methodology's stepwise process throughout the thesis that enabled us to achieve the stated research objectives. It is worth mentioning that the last step is the most influential as it helped us to explore another type of formulation that fueled new ways of tackling the core issues around the dynamic placement and chaining of the VNFs. Needless to say that, the fifth and ongoing work with the master's student benefited most from it.

0.4 Publications and Patents

0.4.1 Publications

The following publications appears in the chronological order of their submission:

1) **Laaziz Lahlou** et al. "FASTSCALE: A fast and scalable evolutionary algorithm for the joint placement and chaining of virtualized services." Journal of Network and Computer Applications 148 (2019): 102429. (IF:5.57);
2) **Laaziz Lahlou** et al. "ARTIMIS: A chemical ReacTion optiMIzation approach for delay-sensitive virtual network Services" (Major revisions submitted to IEEE Transactions on Cloud Computing (IF:4.714);
3) **Laaziz Lahlou** et al. "VALKYRIE: A suite of topology-aware clustering approaches for cloud-based virtual network services." (Under review with IEEE Transactions on Serivces Computing (IF:5.823);
4) **Laaziz Lahlou** et al. "DAVINCI : online and Dynamic Adaptation of eVolvable vIrtual Network services over Cloud Infrastructures." (Under review with Future Generation Computer Systems (IF:6.125));

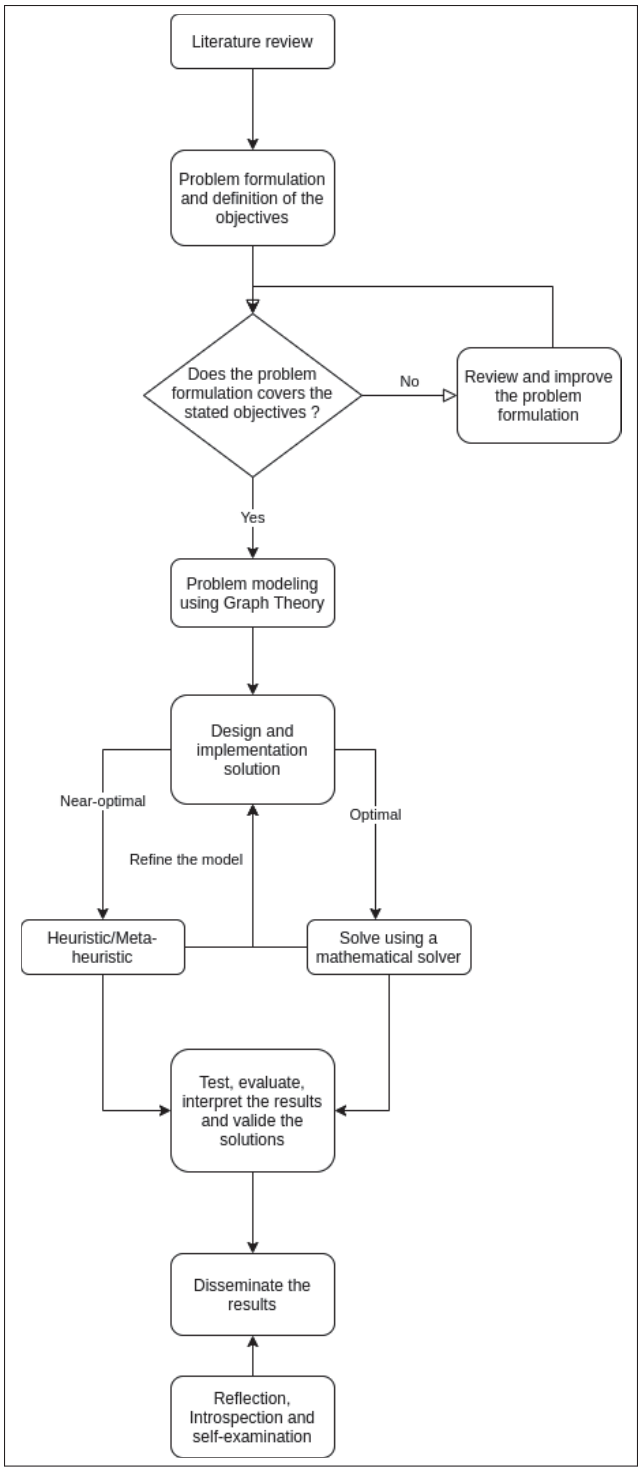


Figure 0.3 The chosen methodology

5) **Laaziz Lahlou** et al. "Cheetah: A fast unsupervised learning technique to provision next generation network services". WOC'20 Collocated with ACM/IFIP Middleware, December 7–11, 2020, Delft, Netherlands;
6) **Laaziz Lahlou** et al. "RAFALE: smaRt and scalable orchestrAtion system For virtuAL network sErvices." (Under review with 2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)).

0.4.2 Patents

1) **Laaziz Lahlou**, Nadjia Kara and Claes Göran Robert Edstom. Joint placement and chaining of virtual network functions for virtualized systems based on a scalable genetic algorithm. Publication Number (WIPO): WO/2020/026142, Ferbuary 2020;
2) **Laaziz Lahlou**, Imane El Mensoum, Fawaz A. Khasawneh, Nadjia Kara and Claes Edstrom. MACHLAREN : unsupervised MACHine Learning techniques for mAnagement of REsources in virtualized and non-virtualized iNfrastructures. Provisional Application for Patent reference number P79104, October 2019.

0.5 Thesis organization

As this work is completed on an articles-basis, we begin with a general literature review, and then we detail each of our papers in a separate chapter. Finally, we conclude the thesis and outline future directions from the latter part's remaining open research issues.

CHAPTER 1

LITERATURE REVIEW

This chapter presents and highlights the most recent approaches proposed in the literature that relate to the dynamic placement and adaptation of service function chains. It is worth mentioning that this chapter is split into two parts where the former discusses the solutions proposed for the dynamic placement and chaining of the VNFs, whereas the latter focuses on the dynamic adaptation of service function chains. In each part, we position our contributions for the existing methods that we consider the closest to our goals in the present thesis.

Notwithstanding, each article includes a thorough literature review on its own, and special mention should be made of the machine learning and deep learning-based approaches that are not included in this chapter since this goes beyond the present thesis's scope.

1.1 Placement and Chaining of the VNFs

A set of resource allocation heuristics and a column generation decomposition technique are proposed by Muhammad, Sorkhoh, Qu & Assi (2021) to optimize IPTV placement multicast services in an NFV-enabled network by satisfying several requirements, including end-to-end delay. However, the proposed heuristics and Dantzig-Wolfe-based method suffer from scalability issues and higher computation time.

Authors in Xie, Ma & Zhao (2020a) explored how the notion of parallelism would reduce the total latency of a service function chain. The proposed work's main idea is to examine the actions that VNFs perform to find the best strategy to run them in parallel with minimum latency. The authors formulated the problem as an integer linear program and proposed a randomized rounding technique with theoretical guarantees for small-scale instances of the problem. For large-scale instances of the problem, a heuristic is proposed and coupled with the parallelism and consolidation strategies. A similar principle is also leveraged by Tashtarian, Zhani, Fatemipour & Yazdani (2020).

An adapted capacitated shortest path tour problem is leveraged to be used as a framework for the placement and chaining of the VNFs and a greedy heuristic algorithm proposed by Sasabe & Hara (2020). The main idea of the proposed approach is to transform the underlying infrastructure to use the particular formulation of the capacitated shortest path tour by adding fictitious nodes capable of hosting the VNFs, which are connected with the physical nodes. However, as the shortest path tour problem assumes a directed acyclic graph, the elaborated method is limited only to linear service function chain topologies.

A grey wolf-based metaheuristic optimization approach is proposed by Xing, Zhou, Wang, Luo, Dai, Li & Yang (2019). The authors studied the minimization of the end-to-end delay for the placement of service function chains and suggested an integer encoding grey wolf optimizer (IEGWO) to solve it. However, the authors considered offline and static versions of the problem, and the focus of their study was on the quality of the obtained solutions and compared them with well-known metaheuristic approaches (e.g., particle swarm optimization, genetic algorithm). Moreover, the proposed technique suffers from high computational complexity, and the evaluated instances contain only linear service function chains. It is essential to mention that minimizing the end-to-end delay, provided as an objective function, of a given service function chain is different than when we search for a feasible solution where the end-to-end delay is considered as a constraint. That is to say, that a solution that minimizes the end-to-end delay would not be necessarily feasible if the end-to-end delay is a constraint to satisfy.

To improve the time-efficiency of placement and chaining algorithms, authors in Qi, Shen & Wang (2019) proposed the notion of accessible scope. The proposed approach's main idea is to reduce the search space's size in terms of servers by considering only those with a smaller distance from the ingress and egress nodes of the infrastructure. Therefore, requests having similar ingress and egress nodes are grouped and placed within that specific accessible scope. However, a close look at the algorithm and the experimental evaluation, the authors did only partition their network into some subnetworks where requests must have the same ingress and egress nodes. Although the authors did reduce the execution time to some extent, the placement and chaining algorithm

is tied to specific settings and could not be generalized. Moreover, the authors investigated only simple service function chain topologies with a limited number of VNFs.

Authors in Xie, Wang & Dai (2019), Xie, Wang & Dai (2020b) studied the dynamic placement and chaining of VNFs in a dynamic setting and formulated it as an Integer Linear Program. The proposed approaches aim to maximize the provider's revenue by admitting many service function chains and minimizing the penalty costs. An approximation algorithm termed DAFT (DynAmic vnF placemenT), based on the primal-dual paradigm is proposed to tackle the computational complexity of the ILP with a provable competitive ratio of (1 - 1/e) to the optimal offline solution. Despite such theoretical analysis, the proposed approach does not consider chaining as part of the optimization. The authors simplified the problem by considering a set of existing paths between each pair of servers as input to the algorithm. Furthermore, neither quality of service metrics are considered (e.g., end-to-end delay) nor complex service function chain topologies. Finally, the authors did not explain how the revenue and the penalty parameters are calculated and set to perform the experiments.

A similar theoretical analysis like in Xie *et al.* (2020b) is conducted by Sang, Ji, Gupta, Du & Ye (2017). The authors proposed two simple greedy algorithms with provable optimality both for general and tree network topologies where the aim is to minimize the number of used VNF instances. However, the proposed algorithms suffer from a major limitation: the model considers only one single VNF. Therefore, there is no evidence about the proposed algorithms' generalization and how they would perform for more than one VNF.

Authors in Kutiel & Rawitz (2019) studied the resource allocation problem in the context of NFV and SDN from a theoretical perspective and proposed a set of approximation algorithms with theoretical guarantees. However, as the paper focuses only on the theoretical guarantees, it lacks experimental evidence to appreciate the proposed solutions.

A set of online and batch algorithms was proposed to minimize the placement and chaining energy consumption in Soualah, Mechtri, Ghribi & Zeghlache (2019). The proposed approach is a unique formulation of the problem, which is suggested with a limited number of candidate hosts

selected, using a Bipartite graph transformation algorithm of the network infrastructure to reduce the Integer Linear Program's convergence time. Furthermore, toward increasing the number of accepted requests, the authors enabled sharing the VNFs between tenants and integrated it into their mathematical model. The way the sharing mechanism works is primarily based on reusing the already deployed VNFs with high capacities in terms of computing resources for which the cumulative requirements of the incoming request should not exceed their limits. However, the authors did not discuss whether the existing paths are shared between the VNFs of the different accepted tenants. Furthermore, the proposed sharing mechanism is included only to improve the acceptance rate and is not thoroughly evaluated to assess its impact on the quality of service of each tenant's request while minimizing the energy consumption.

In the same context as Soualah *et al.* (2019), authors in Sun, Li, Yu, Vasilakos, Du & Guizani (2019) tackled the service function chain orchestration problem in a multi-domain setting to propose an energy-efficient solution. The proposed approach uses a simplified representation of the domains to reduce the search space when seeking a feasible solution for the deployment of the service function chains. Therefore, this technique enables the heuristic to perform relatively with low-computational complexity. However, the authors considered only linear service function chain topologies, whereas the end-to-end delay was not part of the optimization.

Authors in Pei, Hong, Xue & Li (2018) studied the differentiated routing problem for service function chaining in an SDN and NFV-enabled networks. The authors formulated the problem as a Binary Integer Linear Programming model and proposed a Resource Aware Routing Algorithm (RA-RA), which aims to minimize the resource consumption costs of different types of flows belonging to the various requests. The proposed algorithm uses a layered graph that includes several VNFs, composing the request in their order of appearance. The k-shortest path algorithm is then fed with the layered graph to select the least-cost path from the ingress node to the egress node of the request, which traverses all the VNFs. Notwithstanding, the proposed algorithm assumes the VNFs are already deployed and tied to only requests with linear topologies that process the traffic in one-way.

Authors in Trajkovska, Kourtis, Sakkas, Baudinot, Silva, Harsh, Xylouris, Bohnert & Koumaras (2017) proposed Netfloc an open-source SDK (Software Development Kit) coupled with SDN to realize service function chaining. Netfloc comes with a set of libraries that support OpenDaylight as an SDN controller. Moreover, a proof of concept is proposed to test different use-case real-world service function chains. However, the authors did not propose any placement and chaining algorithm, and only linear topologies were tested and analyzed.

A particular study aimed at multi-cast services is conducted by Kiji, Sato, Shinkuma & Oki (2020). However, the main limitation of the proposed solution is its non-scalability aptitude and higher execution time. Similarly, authors in Asgarian, Mirjalily & Luo (2020) proposed an approximate solution and a heuristic that enable finding feasible solutions quickly. However, the end-to-end delay is not part of the optimization model.

Authors in Nguyen, Minoux & Fdida (2019) investigated the design of flow cover inequalities to boost the processing time and improve the quality of the obtained solutions for the placement and chaining of VNFs, which are integrated into standard mathematical solvers. The proposed approach's main objective is to optimize resource utilization (e.g., nodes and links). Notwithstanding, although the flow cover inequalities displayed benefits still the proposed method suffers from higher execution time.

1.2 Service Function Chain Adaptation

Authors in Dominicini, Vassoler, Valentim, Villaca, Ribeiro, Martinello & Zambon (2020) addressed the problem of embedding a service function chain in the network infrastructure to support any topology by extending the concept of network fabric and proposed a prototype termed as KeySFC. The main aim of KeySFC is to integrate chaining and routing decisions to improve the adaptation of service function chains in response to demand (e.g., bandwidth, latency) changes. Also, KeySFC takes advantage of duplicated VNF instances deployed over different servers and path migrations to achieve the goal, as mentioned earlier. In terms of implementation, KeySFC uses OpenStack as a proof-of-concept to showcase the realization of a

service function chain with traffic steering capability in a software-defined network scenario, which takes advantage of source routing to eliminate forwarding tables in core nodes. However, the proposed approach does not suggest any optimization algorithm to work in tandem with the proposed prototype. Furthermore, KeySFC assumes trivial schemes where VNFs are either consolidated in one server or distributed over adjacent servers following their order of appearance in the topology.

Authors in Yi, Wang, Huang & Dong (2019) tackled the issue of migrating VNFs under the assumption that several service function chains use them modeled it as an integer linear program to minimize its influence rather than the migration process itself after executing it. To scale with the size of the problem, i.e., the number of service function chains, a multi-criteria decision technique is suggested using Fuzzy logic by balancing the overall traffic load with the overall delay. It computes the difference in terms of service function chain delay and network load to decide the best location of the migrated VNF subject to the constraint of having a lower network delay. The primary triggers related to the migration of VNFs are when the nodes and links are overloaded during a certain threshold. Notwithstanding, the proposed technique suffers from higher execution time for large infrastructures, and the authors did not evaluate it using a real testbed. Similarly, we are neither testing our solutions in a real testbed.

Authors in Mehmood, Muhammad, Ahmed Khan, Diaz Rivera, Iqbal, Ul Islam & Song (2020) developed a system that can automatically scale VNFs given their resource execution patterns coupled with a threshold mechanism that monitors the level of CPU utilization for better infrastructure energy optimization. The proposed mechanism estimates the number of CPU cores required to be added based on the actions it performs, the minimum number of CPU cores, and the percentage of energy consumption.

An autoscaling and provisioning heuristic is proposed by Son & Buyya (2019) for latency-sensitive service function chains and implemented within a real-life testbed. The resource management technique enables instantiating new VNF instances whenever a given VNF is overloaded and decides where to deploy it to reduce the overall latency. However, the proposed

solution does not optimize the whole service function chain in terms of placement and chaining of the VNFs, considers only the latency, and ignores other compute (e.g., CPU, RAM, and Disk Storage) and network resources (e.g., bandwidth).

Authors in Hejja & Hesselbach (2019) investigated the effects of traffic migration and the consolidation of VNFs on resource allocation algorithms to reduce data center power consumption costs. The study suggests that resource allocation algorithms perform better and quickly when migration is not leveraged. However, the proposed approach is intended only to consolidate the entire service function chain over one server.

Authors in Zou, Li, Wang, Qi & Sun (2018) studied the impact of virtualization technology over the time it takes to migrate VNFs over an SDN infrastructure. The study highlights that implementing network functions over Docker containers helps decrease the migration time significantly (e.g., three to four times) than virtual machines heavier.

1.2.1 Discussions

To the best of our knowledge, all the techniques that use the weighted sum method work only when solving the optimal solution through Integer Linear Programming models. Our careful analysis of each surveyed paper's experimental evaluations and results suggests that most of the existing techniques do not possess the ability to guide the search for a solution with the so indicated weights. Therefore, there is no approach (e.g., greedy algorithms, heuristics, meta-heuristics, dynamic programming, machine learning) where the weights influence the logic (e.g., behavior) behind the process of finding a solution for the placement and chaining of the VNFs.

Most of the proposed approaches assume a strict traversal order for the traffic between a source VNF and a destination VNF. The major drawback of such an assumption is that existing algorithms work only with linear topologies. Thus, the non-scalability aptitude of the methods when it comes to handling complex service topologies.

It is worth mentioning that, in all the following contributions, the VNFs are not assumed to be shared between service function chains like in the reviewed approaches.

1.3 Conlusion

We reviewed in previous sections, existing approaches relevant to the dynamic placement and chaining of the VNFs, and the dynamic adaptation of service function chains using various elasticity mechanisms (e.g., vertical and horizontal scaling). As discussed previously, the proposed approaches present some limitations that must be resolved in order to ensure a placement and chaining of the VNFs and their dynamic adaptation that satisfies the service level agreement contract negotiated with each user and reduce the costs incurred by a use of these.

The following chapters are devoted to tackling the limitations mentioned above and providing solutions proposed in this thesis.

CHAPTER 2

FASTSCALE: A FAST AND SCALABLE EVOLUTIONARY ALGORITHM FOR THE JOINT PLACEMENT AND CHAINING OF VIRTUALIZED SERVICES.

Laaziz Lahlou[1] , Nadjia Kara[1] , Rafi Rabipour[2] , Claes Edstrom[2] , Yves Lemieux[2]

[1] Département de Génie Mécanique, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3

[2] Ericsson, Canada,
8275 Rte Transcanadienne, Saint-Laurent, QC H4S 0B6

Paper published in the Journal of Network and Computer Applications on August 2019
(DOI:10.1016/j.jnca.2019.102429)

2.1 Abstract

Service function chaining or in-line services in network function virtualization is a promising approach for network and service providers as it allows them to dynamically instantiate network functions and interconnect them according to a predetermined policy on-the-fly. It brings flexibility, easy management and rapid deployment of new virtual network services. With the advent of 5G and the concept of network slicing, virtual network services are becoming increasingly complex, not only in terms of topologies, but also in terms of stringent requirements that need to be fulfilled. An optimal deployment of these service function chains, and virtual network services in general, calls for an approach that considers the operational, traffic, and energy costs and QoS constraints jointly from a multi-objective mathematical perspective. In this paper, we propose a multi-objective integer linear program for the joint placement and chaining of virtual network services of different topologies (linear and non-linear) to solve the problem optimally and an evolutionary algorithm (cultural algorithm) for medium and large-scale instances (not limited to 3 to 5 VNFs per SFC). To the best of our knowledge, we are the first to consider this problem from this perspective and propose a solution to it. Extensive experiments demonstrate the effectiveness and efficiency of our proposed evolutionary algorithm versus the optimal solution.

Keywords : Network function virtualization, attributed network infrastructures, clustering, multi-objective optimization, topological structure, attribute similarity.

2.2 Introduction

There is currently a huge shift in how network and cloud services are offered by service and cloud providers to end-users thanks to Network Function Virtualization (NFV); this concept represents a pioneering paradigm and architecture for delivering next-generation network services by decoupling network functions from their hardware to offer them as virtual appliances and deploy them on commodity hardware anytime and anywhere. This allows the rapid composition of virtual network functions (VNFs) (e.g., firewalls, network address translations, intrusion detection systems) in sequence to form a service function chain, in-line service, etc., providing the benefits of on-demand resource allocation, higher level of automation and scaling capabilities to handle traffic fluctuations and very demanding requirements specifically related to SLA. For more information and further reading, we refer the reader to these comprehensive references Mijumbi *et al.* (2016), Yi, Wang, Li, Das & Huang (2018). Thanks to NFV, service providers may offer OPEX and CAPEX virtual services at reduced cost. However, with the advent of 5G ETSI-5G and the concept of network slicing ETSI-NS, virtual network services are becoming increasingly. In fact, network services may comprise an in-line service in addition to network service-based VNFs interconnected together to form a complex virtualized network service topology. Yet, virtualized network services must fulfill different stringent requirements both during the deployment phase and after being deployed following the SLA agreed to between the service provider and clients. Effectively deploying these virtual network services represents a complex and major challenge to overcome. This is particularly the case when competing and conflicting objectives are at stake, in terms of operational and network traffic costs, mainly from a service provider stand point. Note also that this is true both during and after the deployment phases, while considering clients' QoS constraints, e.g., end-to-end delays. To contribute to the minimization of the power consumption one must consider not only the number of active servers, but also their energy consumption, while allocating needed resources to the VNFs for

their proper operation and purposes. It is well known that over 50% of the maximum power consumption by physical machines occurs while they are in idle state Gandhi, Harchol-Balter, Das & Lefurgy (2009). Few works have considered this important aspect Pham, Tran, Ren, Saad & Hong (2020), Bari, Chowdhury, Ahmed & Boutaba (2015) in their formulations. Virtual network functions communicate and exchange traffic according to the topology of the virtual network service and the policy as defined and applied by the service provider. On the one hand, the farther the VNFs are located from one another, the greater the bandwidth they consume, which inevitably results in increased power consumption since more physical links are used, and longer paths are generated, increasing the total delay, and likely resulting in an SLA violation Liu, Li, Zhang, Su & Jin (2017).

On the other hand, putting the VNFs altogether as proposed by Pham *et al.* (2020) using the consolidation strategy may not be realistic, and results in increased power consumption within the same group of servers. In fact, such consolidation may not always even be possible if we add affinity and anti-affinity rules. The strategy reduces the number of operational and active servers, but might not reduce their power consumption. Without consolidation, resources could be underutilized, resulting in a non-energy efficient and a non-cost-effective solution. This fact must therefore be considered, and a tradeoff identified to efficiently use available resources while meeting the objectives of the service provider and the requirements of the clients in terms of Quality of Service (QoS). When selecting the appropriate servers to deploy VNFs, the above aspects, as well as the residual resources of the underlying infrastructure must be considered in order to allow their optimal use and find a resource-efficient solution. We leverage the mathematical expression borrowed from Pham *et al.* (2020), used mainly for finding preference lists, and integrate it into our mathematical model with some adjustments. The aim is to find the best fitting resources among the available servers with respect to their residual resources. The search space will consequently be explored according to this consideration.

In certain cases, the deployment of virtual network services will also call for affinity and anti-affinity rules to meet particular objectives, including performance, reliability and security. However, in their work, the authors in Addis, Belabed, Bouet & Secci (2015) considered only

affinity and anti-affinity rules specifically from the ILP formulation, and did not discuss an implementation from their heuristic perspective, while others Ayoubi, Chowdhury & Boutaba (2018), Alleg, Ahmed, Mosbah, Riggio & Boutaba (2017), Khebbache, Hadji & Zeghlache (2018), Pham *et al.* (2020), Khebbache, Hadji & Zeghlache (2017), Luizelli, Bays, Buriol, Barcellos & Gaspary (2015a), Luizelli, da Costa Cordeiro, Buriol & Gaspary (2017), Ghribi, Mechtri & Zeghlache (2016), Mechtri, Ghribi & Zeghlache (2016), Bari *et al.* (2015) did not consider affinity and anti-affinity in their respective works. It is worth mentioning that none of these works Ayoubi *et al.* (2018), Alleg *et al.* (2017), Pham *et al.* (2020), Khebbache *et al.* (2017), Luizelli *et al.* (2015a), Mechtri *et al.* (2016), Addis *et al.* (2015), Bari *et al.* (2015) have considered the aforementioned challenges and competing objectives in the context of a multi-objective mathematical approach within the proposed solutions, the integration of affinity and anti-affinity rules within the proposed heuristic solutions, and the complex topologies that may exist for a virtual network service.

Given the nature of this challenging problem, we propose to tackle it from a multi-objective mathematical perspective with four competing objective functions in order to minimize the associated operational, traffic and energy costs, while satisfying the client requirements in terms of QoS constraints. Therefore, it is important to highlight that neither the presented mathematical formulation nor the elaborated metaheuristic approach, with the aim of performing the placement and chaining of the VNFs in one step, proposed in this paper, were addressed by Ayoubi *et al.* (2018), Alleg *et al.* (2017), Khebbache *et al.* (2018), Pham *et al.* (2020), Khebbache *et al.* (2017), Luizelli *et al.* (2015a), Luizelli *et al.* (2017), Ghribi *et al.* (2016), Mechtri *et al.* (2016), Addis *et al.* (2015), Bari *et al.* (2015) and Liu *et al.* (2017). Moreover, to the best of our knowledge, they have not been addressed in the literature.

Our aim in this paper is to tackle the problem of the placement and chaining of virtualized network services which are composed of several virtualized network functions (VNFs), taking into consideration their dependencies as well. Figure 2.1, however, shows a scenario of a virtualized network service having a service function chain, composed of a firewall, an intrusion detection and a prevention system, as well as a network address translation; further, it has IMS

(IP Multimedia Subsystem)-based virtualized network functions, which we will simply refer to as a network service going forward. Also, the terms in-line service and service function chain are used interchangeably herein, and pertain to the same concept.

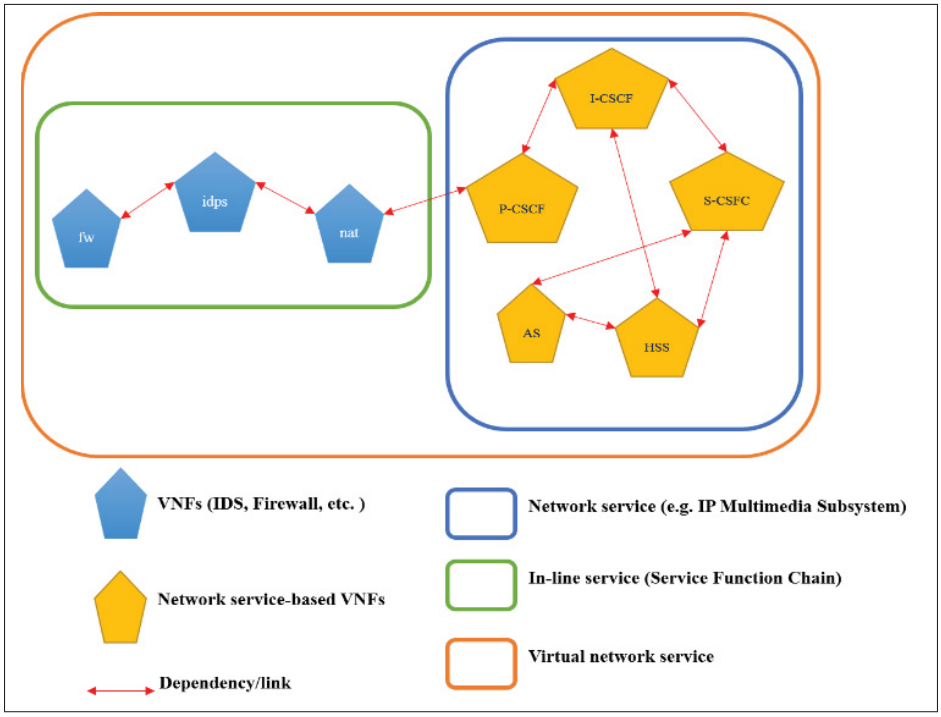


Figure 2.1 An example of a network scenario that embodies a network/service provider with a set of VNF licenses

In this paper, we fill this gap in the research literature with the following main contributions:

1) We provide the first formulation of the joint placement and chaining of VNFs for virtual network services, while not limiting our scope to in-line services with up to three to four VNFs and linear topologies;
2) The problem is formulated as an Integer Linear Program, implemented and solved in line with Gurobi, to find optimal solutions for small-scale networks;
3) We propose FASTSCALE, a fast and scalable multi-objective metaheuristic-based approach to handle medium and large-scale instances of the problem, leveraging the same ILP structure as in 2) for reliable benchmarking;

4) FASTSCALE's performance in terms of solution quality and scalability is assessed using real-world topologies: small enterprise and data center networks.

The rest of the paper is arranged as follows. We review relevant works in section 2.3 and discuss the definitions and methodology adopted in Sections 2.4. Next, we present the theoretical foundations and problem statement in Section 2.4.1. In Section 2.5, we present the ILP (Integer Linear Program) solution. The metaheuristic-based approach is presented in Section 2.6, followed by a performance evaluation in Section 2.7. Finally, Section 2.8 concludes this paper and identifies avenues for future research.

2.3 Related Works

Since its instigation by ETSI, service function chaining and VNF placement have attracted several actors within the context of NFV, both from academia and industry. This is a promising paradigm shift since it is proposed as a solution for reducing both capital (CAPEX) and operational expenditures (OPEX). However, to fully exploit the benefits of this technology enabler, several challenges need to be addressed. Methods and models must be brought in to tackle the problem from the perspectives of the service provider, the network operator and the end-user, with different service level agreements and stringent requirements. Several solutions and formulations have been proposed to optimize different objectives.

Khebbache *et al.* (2018) proposed a bi-objective genetic algorithm for VNF chain placement. Their algorithm is solved using NSGA-II and aims at minimizing both the number of servers and links used. However, their encoding scheme does not allow the joint placement and chaining of VNFs since it captures only the placement of VNFs and ignores their dependencies. In addition, QoS constraints, in terms of end-to-end delays, are not considered. Moreover, evaluation shows that this topology is limited to linear dependencies between the VNFs.

Ayoubi *et al.* (2018) addressed the service function chain orchestration problem from a flexibility and virtual network function reordering perspective. Their aim was to study the impact on resource allocation, and they proposed Khaleesi to minimize bandwidth consumption. However,

the scope of this work is limited to a very specific type of service function chain since the assumption of swapping the order of two VNFs would not affect their internal state. This is not always the case since network services are most often driven by policies.

Ghribi *et al.* (2016) applied a dynamic programming approach for the joint placement and chaining of VNFs with the aim of minimizing placement costs and thus reduce the number of servers used. However, the proposed solution is limited to linear topologies since the recurrence relation is not generic, and can thus not handle complex topologies. Additionally, it does not consider QoS constraints, and the only resources required for the VNFs are CPUs.

A couple of solutions have been proposed by Pham *et al.* (2020) to enable joint operational and network traffic cost minimization for VNF placement in service function chains. A Markov Approximation technique (MA) was proposed for coping with the combinatorial complexity of the problem. Due to its long convergence time, the authors suggested SAMA, which merges MA with matching theory. This approach works iteratively by first selecting a subset of servers, then placing the VNFs to reduce the cost functions. Both centralized and distributed variants of SAMA are devised, relying on game theory and a consolidation strategy. However, the proposed solutions are centered on a consolidation strategy by packing the VNFs of each service function together within the same server. This is not realistic if we consider affinity and anti-affinity constraints, which would then result in conflicts, thereby limiting the scope of these solutions. In addition, no QoS were considered and the focus was on linear topologies.

Luizelli *et al.* (2015a) proposed a virtual network function placement and service function chaining approach, with the objective of minimizing the number of virtual network function instances, since the authors suppose that several copies of such instances are available, with respect to constraints such as end-to-end delay and processing time per amount of traffic. To handle large infrastructures, a heuristic using the binary search method was proposed to find the lowest number of network functions needed, which were then integrated into the solver to reduce the search space for a quick convergence towards optimal solutions. However, this approach suffers from significant CPU and bandwidth overheads as VNFs are shared between tenants.

Moreover, the authors addressed only in-line services. An improved version of the heuristic was proposed in Luizelli *et al.* (2017), where they use a variable neighborhood technique in order to find best quality solutions and explore the search space efficiently for large infrastructures, but with no improvements seen in terms of runtime, which remained high.

A hybrid approach combining both virtual network functions and physical appliance placement for service function chaining is proposed by Mechtri *et al.* (2016). The approach is based on the Eigen-decomposition technique used in Algebra. The virtual network forwarding graph mapping into the substrate physical network is solved by an extended version of Umeyma's Eigen-decomposition technique for the graph matching problem. However, in order to use the method, a pre-processing which involves adjusting the size of service requests to fit the size of the network infrastructure must be completed. Moreover, the QoS related constraint is not considered in the evaluations. Also, the authors did not discuss how a hybrid service function placement could be realized in a realistic environment.

Khebbache *et al.* (2017) addressed the problem of virtual network function placement and chaining and proposed two approaches. The first transforms the service function chain request into a graph containing cycles Gibbons. This transformation involves adding a fictitious edge connecting the first and last virtual network functions within the graph. Doing so, the authors find subsets of cycles in the physical network that can host the virtual network functions that meet their requirements. Secondly, for scalability issues, the authors provided a two-stage matrix-based algorithm. The authors realized the placement of virtual network functions in the first stage, and then, the chaining in the second stage. This however, does not lead to efficient solutions as more requests could be rejected and resources wasted. Similarly, to Pham *et al.* (2020), they also employ the consolidation strategy to pack as many VNFs as possible within the same server. However, it does not consider QoS constraints in terms of end-to-end delay.

Addis *et al.* (2015) formulated the problem of virtual network function placement and routing optimization for virtual Customer Premises Equipment use cases ETSI as a mixed integer linear program. The main objective of their work was to minimize the number of CPU cores used by

the virtual network functions and the network bandwidth at the same time. In this approach, the authors considered the case where traffic load changes and undergoes compression and decompression, using special network functions along the service function chain to decrease latency. However, this approach is tied to a specific service function chain with a special use case.

Similar to Luizelli *et al.* (2015a), Bari *et al.* (2015) propose an approach that combines two techniques, namely, multistage graph and Viterbi-based dynamic programming, aimed at minimizing the operational costs of the network. However, this approach assumes that only CPU resources are needed by the VNFs. Also, the topologies of the in-line services are assumed to be linear, and not exceeding 4 VNFs.

Contrary to the aforementioned works, Alleg *et al.* (2017) addressed the problem from a different perspective by investigating the impact of flexible resource allocation to a given VNF on its latency, with the aim of avoiding allocation of unnecessary resources. The aim was to understand the relationship between allocated resources and VNF performances. However, the authors presented only Integer Linear Progam solutions, which cannot be used for real deployments.

Now, we will briefly discuss the main aspects that distinguish our approach from the solutions proposed above. Unlike Ayoubi *et al.* (2018), Alleg *et al.* (2017), Pham *et al.* (2020), Khebbache *et al.* (2017), Luizelli *et al.* (2015a), Luizelli *et al.* (2017), Ghribi *et al.* (2016), Mechtri *et al.* (2016), Addis *et al.* (2015), Bari *et al.* (2015), Liu *et al.* (2017), we adopt a multi-objective approach instead of a single-objective method in order to deal with the conflicting cost functions, namely, operational cost, traffic cost and energy cost, since the problem is multi-objective in nature. Thus, the solution we obtain, given the constraints, yields the best tradeoff among the goals set by the service provider, which may not be possible if we were to rely on dynamic programming, heuristic or greedy approaches. Moreover, our approach minimizes operational costs similarly to Khebbache *et al.* (2018), Pham *et al.* (2020), Luizelli *et al.* (2015a), Ghribi *et al.* (2016), Bari *et al.* (2015), bandwidth consumption Ayoubi *et al.* (2018), Khebbache *et al.* (2018), Bari *et al.* (2015) and energy costs Pham *et al.* (2020), Bari *et al.* (2015). However,

we do not share VNFs between tenants like Luizelli *et al.* (2015a) since service requests have different requirements both in terms of resources and SLA.

Regarding the topologies, the problem formulated by Ayoubi *et al.* (2018), Alleg *et al.* (2017), Khebbache *et al.* (2018), Pham *et al.* (2020), Khebbache *et al.* (2017), Luizelli *et al.* (2015a), Luizelli *et al.* (2017), Ghribi *et al.* (2016), Mechtri *et al.* (2016), Addis *et al.* (2015), Bari *et al.* (2015), Liu *et al.* (2017) do not consider complex topologies, since they are limited only to three to four VNFs, that is, to linear topologies. In addition, the placement and chaining of VNFs is done in two separate steps Ayoubi *et al.* (2018), Alleg *et al.* (2017), Khebbache *et al.* (2018), Khebbache *et al.* (2017), Luizelli *et al.* (2015a), Luizelli *et al.* (2017), Liu *et al.* (2017). Although Khebbache *et al.* (2018) proposed a bi-objective genetic algorithm, they did not leverage the abstraction provided by the genotype encoding mechanism to consider complex topologies and their dependencies.

Our genotype encoding scheme allows us to handle complex topologies since we consider virtual links within the genotype, together with the affinity and anti-affinity features that might exist between the VNFs and the placement and chaining in one step.

We do not use specific VNF operations comparable to a compression to reduce latency Addis *et al.* (2015), but instead, we consider it as the end-to-end delay to be fulfilled. Yet, compared to Khebbache *et al.* (2017), Mechtri *et al.* (2016), Bari *et al.* (2015), our approach does not apply any pre-treatment on the graph representing the service requests. Like Alleg *et al.* (2017), Addis *et al.* (2015), Bari *et al.* (2015), we consider QoS as well. The virtual network function performance versus resource allocation scheme proposed by Alleg *et al.* (2017) is left for future work in our approach.

Hereunder, we synthesize and position our work in line with the most relevant approaches found in the literature. Moreover, our work is mainly influenced by the aforementioned works and considers their limitations. To the best of our knowledge, we are the first to look at this problem from a broader perspective, i.e., using an approach not limited to in-line services. Table 2.1

summarizes the main differences between our proposed approach and the most relevant existing approaches.

Table 2.1 A comparison between our approach and the state of the art

Solutions	**Criteria**			
	Joint placement and chaining of the VNFs	**QoS constraints**	**Type of services**	**Proposed methods**
Ayoubi *et al.* (2018)	No	No	In-line services	Integer linear program and a heuristic
Alleg *et al.* (2017)	No	Yes	In-line services	Integer linear program
Khebbache *et al.* (2018)	No	No	In-line services	Metaheuristic-based genetic algorithm
Pham *et al.* (2020)	No	No	In-line services	Integer linear program and a set of heuristics
Khebbache *et al.* (2017)	No	No	In-line services	Integer linear program and a matrix-based heuristic
Luizelli *et al.* (2015a, 2017)	No	Yes	In-line services	Integer linear program and a heuristic
Ghribi *et al.* (2016)	Yes	No	In-line services	Dynamic programming
Mechtri *et al.* (2016)	Yes	No	In-line services	Matrix-based heuristic and a greedy algorithm
Liu *et al.* (2017)	No	Yes	In-line services	Integer linear program and a heuristic
Addis *et al.* (2015)	No	Yes	In-line services	Integer linear program
Addis *et al.* (2015)	No	Yes	In-line services	Integer linear program and a heuristic.
Ours	**Yes**	**Yes**	**Virtual network services including in-line services**	**A multi-objective integer linear program and a metaheuristic-based genetic algorithm**

2.4 Definition and Methodology

We will now clarify some terms used in this paper in order to help the reader appreciate our real contribution and situate our work from a high-level perspective.

1) **In-line service or service function chain**: a set of VNFs deployed to process traffic according to a given policy. A typical service function chain or in-line service comprises middleboxes such as a firewall, a proxy, deep packet inspection, etc.;
2) **Network service-based VNFs**: a set of virtualized network functions specific to a service like IMS (IP Multimedia Subsystem), CDN (Content Delivery Network), etc. For example, IMS consists of a set of virtualized network functions such as P-CSCF, I-CSCF, S-CSCF, HSS, etc.;
3) **Affinity**: a rule that establishes a relationship between a group of VNFs to be collocated within the same space, i.e., a server or a data center;
4) **Anti-affinity**: a rule stating that a group of VNFs cannot share the same space (i.e., a server to be used for security or reliability purposes);

5) **Virtualized network service (VNS)**: a virtual network service that comprises an in-line service and a set of networks service-based VNFs. Figure 2.1 depicts this definition and situates our work with respect to the state of the art.

The problem, as depicted in Figure 2.1, is modeled thanks to the Integer Linear Program mathematical framework, with the aim of providing an optimal solution similar to most of the existing solutions in the literature Ayoubi *et al.* (2018), Alleg *et al.* (2017), Pham *et al.* (2020), Khebbache *et al.* (2017), Luizelli *et al.* (2015a), Mechtri *et al.* (2016), Addis *et al.* (2015), Bari *et al.* (2015). However, we adopt a multi-objective mathematical approach instead of a weighted sum approach with tunable parameters, since a weighted sum approach fails to find a tradeoff between the different cost functions involved in the model Gen & Runwei Cheng (1999). The computational complexity of service function chain problems has been demonstrated to belong to the NP-Complete class Addis, Gao & Carello (2018). They are thus hard to solve within a reasonable time frame, given their nature and number of instances that are usually given as inputs. In fact, these models are solved only for a few instances of the problem using commercially available solvers such as Gurobi, CPLEX IBM, Mosek ApS, etc. In our work, we use Gurobi to solve the model.

When the problem scales to medium and large instance numbers, it is common to rely on different approaches, including heuristics, dynamic programming, greedy algorithms, etc. However, for a problem that needs to be solved from a multi-objective perspective, these approaches are not well-suited. This is because they are not designed for this purpose, and the search space is difficult to handle in a flexible and efficient manner Brownlee (2011). Thus, we opt for a metaheuristic or Cultural Algorithm with a Genetic Algorithm (Cultural Genetic Algorithm - CGA) embedded in its core to overcome the limitations of heuristics, greedy and dynamic programming solutions and its ability to interact and handle the search space in an efficient manner Gen & Runwei Cheng (1999).

2.4.1 System Model and Problem Statement

In this section, we start by providing formal definitions of physical and virtual network services and listing the terms pertaining to the scope of our work. Then, we present the problem statement before diving into the mathematical model used both for the ILP solution and the metaheuristic-based approach. The variables used by the mathematical model with their meanings are summarized in Table 2.2.

Table 2.2 List of terms and their meanings

Terms	Meanings
V_p	The set of servers
V_v	The set of virtual network functions
E_p	The set of physical links connecting the servers
E_v	The set of virtual links
G_p	The physical network topology (V_p, E_p)
G_v	The virtual network topology (V_v, E_v)
VNS	The set of Virtual Network Services
$Bw^{(u,v),vns}$	Bandwidth required by virtual link (u, v) belonging to virtual network service VNS
$\beta^{(u,v),vns}_{(x,y)}$	Decision variable for the chaining of VNFs
$\alpha^{vns}_{(u,x)}$	Decision variable for the placement of VNFs
$D_{x,y}$	Distance between server x and y
$C^{(x,y)}$	The capacity of physical link (x, y)
R^p_u	Amount of resource type p (CPU, Memory and Storage) required by VNF u
C^p_x	Amount of residual resource type p (CPU, Memory and Storage) on server x
$\theta^{u,v}_{max}$	The maximum allowed end-to-end delay for evry virtual link (u, v)
$\Delta_{(x,y)}$	The delay on the physical link (x, y)
Θ_p	The weight (importance) of a resource of type p (CPU, Memory and Storage)
Λ	The monetary weight of transporting one unit of traffic through one physical hop distance of the hosted VNFs
Γ_x	The monetary weight to operate a server x
ϵ	The monetary weight per unit of power consumption
P^{idle}_x and P^{max}_x	The idle and peak power consumption of a server x in kilowatts

2.4.2 Physical Network

The physical network is represented as an undirected graph $G_p = (V_p, E_p)$, where V_p is the set of servers and E_p is the set of links interconnecting them. We define a weight function $\nabla_{(x)}$ that associates for each server x five attributes, namely, the number of CPU cores, memory size,

storage capacity and power consumption at idle and peak states. We leverage the same weight function $\nabla(l)$ to associate two attributes for each physical link l, the delay and link capacity. We denote these attributes as the available resources on each server and each physical link.

2.4.3 Virtual Network Services

Akin to the physical network, the virtual network service is represented as an undirected graph $G_v = (V_v, E_v)$, where V_v is the set of VNFs and E_v is the set of virtual links or dependencies between VNFs forming a complex topology. We define a weight function $\tau(v)$ that associates for each VNF three attributes, namely, the number of CPU cores, Memory and Storage capacity. Similarly, for each virtual link connecting two VNFs, $\tau(vlink)$ associates the maximum allowed end-to-end delay and the required bandwidth. These attributes are denoted as the required resources by the virtual network service.

2.4.4 Problem Statement

Given a set of VNS requests and a network infrastructure, the aim is to:

1) Minimize the operational, traffic and energy costs;
2) Find the best-fit resources for the VNFs;
3) Subject to the following constraints:

- Network infrastructure resources should not be overcommitted;
- QoS in terms of end-to-end delay have to be fulfilled;
- The placement and chaining of the VNFs should be done in one-step.

2.5 ILP Formulation

In this section, we present our mathematical model, following the Integer Linear Programming framework, for the joint placement and chaining of VNFs.

2.5.1 Decision Variables

The placement of a VNF belonging to a virtual network service is realized by the following decision variable:

$$\alpha_{u,x}^{vns} = \begin{cases} 1, & \text{if VNF } u \text{ of a } vns \text{ is placed on server } x \\ 0, & \text{otherwise} \end{cases}$$

The following decision variable determines the chaining by taking every consecutive pair of VNFs:

$$\beta_{(x,y)}^{(u,v)} = \begin{cases} 1, & \text{if physical link } (x, y) \text{ can carry the traffic of VNFs } u \text{ and } v \\ 0, & \text{otherwise} \end{cases}$$

2.5.2 Constraints

$$\sum_{vns} \sum_{u} \alpha_{u,x}^{vns} \times Req_u^{CPU} \leq Rem_x^{CPU}, \forall x \in V_p \tag{2.1}$$

$$\sum_{vns} \sum_{u} \alpha_{u,x}^{vns} \times Req_i^{Memory} \leq Rem_x^{Memory}, \forall x \in V_p \tag{2.2}$$

$$\sum_{vns} \sum_{u} \alpha_{u,x}^{vns} \times Req_u^{Storage} \leq Rem_x^{Storage}, \forall x \in V_p \tag{2.3}$$

$$\sum_{vns} \sum_{(u,v)} \beta_{(x,y)}^{(u,v),vns} \times Bw^{(u,v),vns} \leq C^{(x,y)}, \forall (x, y) \in E_p \tag{2.4}$$

$$\sum_{(x,y)} \beta_{(x,y)}^{(u,v),vns} \times D_{(x,y)} \leq \theta^{(u,v),max}, \forall (u,v) \in E_v, \forall vns \in VNS \tag{2.5}$$

$$\alpha_{u,x}^{vns} + \alpha_{v,y}^{vns} \leq \beta_{(x,y)}^{(u,v),vns} + 1, \forall (u,v) \in E_v, \forall (x,y) \in E_p, \forall vns \in VNS \tag{2.6}$$

$$\alpha_{u,x}^{vns} + \alpha_{v,y}^{vns} + \beta_{(x,y)}^{(u,v),vns} \leq 2, \forall (u,v) \in E_v, \forall (x,y) \in E_p, \forall vns \in VNS \tag{2.7}$$

$$\sum_{y} \beta_{(x,y)}^{(u,v),vns} - \sum_{y} \beta_{(y,x)}^{(u,v),vns} = \alpha_{u,x}^{vns} - \alpha_{v,x}^{vns}, \forall (u,v) \in E_v, \forall x \in V_p, \forall vns \in VNS \tag{2.8}$$

$$\sum_{x} \alpha_{u,x}^{vns} = 1, \forall u \in V_v, \forall vns \in VNS \tag{2.9}$$

Constraints (2.1), (2.2) and (2.3) state that server resources, in terms of CPU, Memory and Storage, are not overcommitted. Constraint (2.4) states that the amount of traffic circulating along the virtual links does not exceed the capacity of the physical links. Constraint (2.5) states that the end-to-end delay of each virtual link of each virtual network service is within the maximum allowed delay. This constraint enables to compute the end-to-end delay for linear and non-linear service function chain topologies. Indeed, for a linear topology, the end-to-end delay represents the sum of delays of each virtual link connecting a pair of VNFs composing an SFC. However, for a non-linear topology, the end-to-end delay computation is more complex. Therefore, each delay of a virtual link has to be met in order to guarantee the end-to-end delay of an SFC. An example of how to model this end-to-end delay by considering the delay between the VNF pairs of a service chain is given in Duan (2018). Constraint (2.6) guarantees the joint placement and chaining of VNFs for each virtual network service. Constraint (2.7) states that no physical link should be allocated to a pair of VNFs deployed on the same host for each virtual network service. A path between two VNFs that are placed on different servers that are not adjacent to each other (i.e., linked through a set of switches) is created using constraint (2.8) to assist constraint (2.6) in the joint placement and chaining of the VNFs. Constraint (2.9) ensures that a VNF instance is deployed only on one host.

2.5.3 Objective Functions

We adopt a multi-objective approach with four conflicting objective functions, namely, F1, F2 F3 and F4.

2.5.3.1 Objective function F1

$$\mathrm{F}_1 = Min[\sum_{vns}[\sum_{u}\sum_{x}\alpha_{u,x}^{vns} \times \Gamma_x]] \tag{2.10}$$

F_1 stands for the minimization of the total operational cost, which is a function of the number of VNFs and Γ_x.

2.5.3.2 Objective function F2

$$\mathrm{F}_2 = Min[\sum_{vns}\sum_{(x,y)} D_{(x,y)} \sum_{(u,v)} \Lambda \times Bw^{(u,v),vns}\beta_{(x,y)}^{(u,v)}] \tag{2.11}$$

F_2 stands for the minimization of the total cost to transport traffic of the accepted virtual network services through physical links. It is a function of the distance $D_{(x,y)}$, $Bw^{(u,v),vns}$ and Λ.

2.5.3.3 Objective function F3

$$\mathrm{F}_3 = Min[\sum_{vns}[\sum_{u}\sum_{x}\alpha_{u,x}^{vns}\sum_{p}\Theta_p[1 - \frac{R_u^p}{C_x^p}]^2]] \tag{2.12}$$

Since the VNFs of virtual network services might have different requirements for various resources such as CPU, Memory and Storage, objective function F_3 finds the best-fit resources across the three dimensions (CPU, Memory and Storage). The smaller the value of F_3, the more suitable it is for the servers to be chosen to host the VNFs. Yet, F_3 speeds up the integer linear program. F_3 is taken from Pham *et al.* (2020). However, we adapted it and generalized it for our purpose.

2.5.3.4 Objective function F4

$$\mathrm{F}_4{=}Min\epsilon \times [\sum_{vns}[\sum_{u}\sum_{x}\alpha_{u,x}^{vns}(P_x^{idle} + (P_x^{max} - P_x^{idle}) \times \frac{R_u^{CPU}}{C_x^{CPU}})]] \tag{2.13}$$

F_4 stands for the minimization of the total energy cost (Lee & Zomaya (2012)), which is a function of the idle and max power consumption of servers, CPU utilization and ϵ.

2.6 Cultural Genetic Algorithm

The ILP solution cannot be used for real deployments of virtual network services, given that it does not scale with problem size and is computationally expensive. It is only suited for small instances and benchmarking purposes. Therefore, we propose a metaheuristic-based approach, also known as the Cultural Genetic Algorithm (CGA), which can handle medium and large-scale instances of the problem. Nevertheless, we leverage the same mathematical structure of our ILP solution for the metaheuristic. The intuition behind is to perform reliable benchmarking of CGA against ILP. The main objective of using cultural genetic algorithms Holland (1992) is to improve the embedded search technique of typical genetic algorithms, i.e., crossover, mutation and selection mechanisms. This is achieved by relying on the concept of cultural level, in addition to the population level. Cultural level is implemented through the concept of belief spaces which can be seen as a knowledge base recording the best candidate solutions. We use the situational knowledge to store the overall best individuals throughout the evolutionary process, to be used in the selection operation. The normative knowledge stores the top individuals in terms of fitness value, to be used in the crossover and mutation operations. Finally, history knowledge keeps track of the changes within the network infrastructure.

2.6.1 Initializing a Population

A population can be initialized either deterministically or randomly Gen & Runwei Cheng (1999). A deterministic initialization calls for using certain methods such as first-fit heuristic or even the ILP solution, as it starts by generating initial solutions and iterating through them before reaching the optimal one by implementing the Simplex algorithm. However, this can

impact the overall runtime and algorithmic complexity of the genetic algorithm. Therefore, we chose to initialize the chromosomes randomly.

2.6.2 Genotype Encoding

We leveraged the flexibility and abstraction, provided by genetic algorithms, to design a custom genotype encoding tied specifically to the joint placement and chaining of virtualized network services. The key idea was to search for a mapping between virtual links and tuples. A tuple could be either a direct link or a path connecting two VNFs. By encoding the virtual links within the locus level, we performed the joint placement and chaining of VNFs in one step. Figure 2.2 shows a high-level overview of our genotype encoding for the joint placement and chaining of VNFs integrating affinity and anti-affinity features.

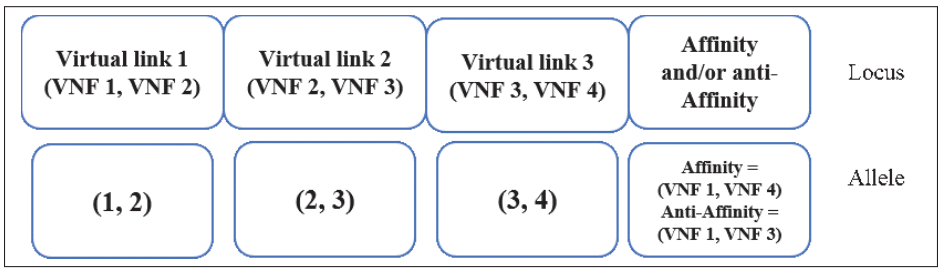


Figure 2.2 Custom-made genotype encoding scheme for solution representation

2.6.3 Genetic Operators

Genetic operators are part of an evolutionary algorithm as they are the mechanisms in charge of evolving the population along the evolutionary process Gen & Runwei Cheng (1999).

2.6.3.1 Crossover operator

A crossover operator is used to vary the population in order to explore several search areas. We adopted the 1-cut point binary crossover operator Gen & Runwei Cheng (1999), which takes

two chromosomes, i.e., parents, and produces two new chromosomes, i.e., children. Basically, a 1-cut point, randomly chosen from the size of the chromosome, is applied to the two parents to split them into two parts. Then, the first child is produced by merging the left part of the first parent with the right part of the second parent. The second child is made following the same principle, except that it starts with the right part of the first parent.

2.6.3.2 Mutation operator

Mutation operators are principally integrated into genetic evolutionary algorithms for exploitation purposes. The aim is to introduce some perturbation on the chromosomes to ensure diversity. Our cultural genetic algorithm makes use of a swapping mutation operator Gen & Runwei Cheng (1999). In a nutshell, this unary operator takes as input a chromosome and two random gene positions. Then it swaps the gene positions and returns the resulting chromosome. However, other mutation and crossover operators can be used and adapted according to our genotype encoding, if necessary. Notwithstanding this, we chose these operators since they do not need additional tunings and custom adaptations.

2.6.4 Fixing Algorithm

Given the custom-made nature of the genotype encoding scheme we designed and the fact that the chromosomes are to be assessed through a fitness evaluation, checking the validity of chromosomes is a critical step. Therefore, we designed GenoFix (Genotype Fixing), an algorithm that checks and fixes genotypes before their evaluation. Hereunder, we present the full description of the algorithm in its high-level form (Algorithm 2.1-2.2). The algorithm works in four steps. The first step, called Initialization, consists in initializing three variables, namely, X_L, X_R, X_I, to record the virtual links having either a same left location, right location or common location (from line 1 to line 4). The second phase consists in decoding the chromosome. In this phase, we start decoding the chromosome, meaning we extract the information present in each gene, i.e., the locations of the VNFs within each corresponding tuple.

Algorithm 2.1 GenoFix (a)

Input: Chromosome

Output: Fixed chromosome

```
1  Initialization
2  X_L ={ } //store the left end points of the virtual links
3  X_R ={ } //store the right end points of the virtual links
4  X_I ={ } //store the virtual links having common endpoint
5  //Decoding the chromosome
6  for each vLink in Chromosome do
7      x_l =[]
8      x_r =[]
9      x_i =[]
10     for each v in Chromosome do
           /* Start with left endpoints                          */
11         if (v and vLink) have similar left endpoint then
12             x_l.append(v)
13             X_L[vLink] = x_l
14         end if
           /* Moving to right endpoints                          */
15         if (v and vLink) have similar left endpoint then
16             x_r.append(v)
17             X_R[vLink] = x_r
18         end if
           /* Moving to common endpoints                         */
19         if (v and vLink) have common endpoint then
20             x_i.append(v)
21             X_I[vLink] = x_i
22         end if
23     end for
24 end for
```

Algorithm 2.2 GenoFix (b)

```
1  (Affinity and Anti-Affinity algoroithm)
   /* Fixing the chromosome                                          */
2  for each vLink in Chromosome do
       /* Start with the vLinks having the same left endpoint        */
3      if (vLink in X_L) then
4          for each element in X_L[vLink] do
5              Index = GetLeftLocation(vLink)
6              SetSameLocation(element)
7          end for
8      end if
       /* Move to the vLinks having the same right endpoint          */
9      if (vLink in X_L) then
10         for each element in X_L[vLink] do
11             Index = GetRightLocation(vLink)
12             SetSameLocation(element)
13         end for
14     end if
15     if (vLink in X_L) then
16         for each element in X_L[vLink] do
17             Index = GetRightLocation(vLink)
18             SetSameLocation(element)
19         end for
20     end if
21     for each vLink in Chromosome do
22         Start = GetLeftLocation(vLink)
23         End = GetRightLocation(vLink)
24         Path = GetPathUsingDijkstra(Start, End)
25         SetPath(vLink, Path)
26     end for
27 end for
```

Then, we store all the virtual links having similar left/right and common endpoints. Two virtual links, l1 and l2, are said to have a common endpoint if the right endpoint of l1 is the same as the left endpoint of l2 (from line 6 to line 24). The third phase consists in executing the affinity and anti-affinity algorithm 2.3, given that we have either affinity constraints, anti-affinity constraints, both affinity and anti-affinity constraints, or even no constraints (line 1 (Algorithm 2.2)).

The last phase consists in fixing the chromosome. We check and match every virtual link within the chromosome with those present in the variables we defined in the initialization phase. The GetRightLocation and GetLeftLocation functions get the right, left and same location of the VNFs within the tuple, respectively, and SetSameLocation sets the identical location of the VNFs. Finally, we leverage Dijkstra's algorithm for chaining the VNFs located at different points using the GetPathUsingDijkstra and SetPath functions (from line 2 to line 27 in Algorithm 2.2). It is worth mentioning that we used priority queues for the implementation of Dijkstra's algorithm in order to reduce its algorithmic complexity to . To sum up, it should be noted that the joint placement and chaining of VNFs is done at the genotype level and is integrated within the GenoFix algorithm, which is the salient feature of our contribution.

2.6.5 Affinity and Anti-Affinity Algorithm

Now, we describe the algorithm we designed for the affinity and anti-affinity constraints integrated into our custom-made genotype (Algorithm 2.3). Basically, this algorithm has two steps. We start with the affinity constraint, followed by the anti-affinity constraint. The rationale behind this order is driven by the fact that the anti-affinity constraint might conflict with the affinity constraint, i.e., the whole structure of the chromosome may turn out to be invalid because of the location constraints. We begin by decoding the "slice" of the chromosome having the affinity constraint (from line 1 to line 4). Then, we go through all the virtual links within the chromosome and check if the IDs of VNFs within the affinity constraint appear in it. If the IDs appear, then we set them to the same location. This is done by checking both the left and right endpoints of the virtual links (from line 5 to line 14). Next, the same logic as above is followed for the anti-affinity constraints (from line 15 to line 31).

2.7 Evaluation

In this section, we evaluate our proposed solutions, specifically, the ILP solution and FASTSCALE. The ILP solution is assessed only on a small enterprise network instance due to the longer

Algorithm 2.3 Affinity and Anti-Affinity Algorithm

```
Input: Chromosome
Output: Chromosome

Affinity = GetAffinity(Chromosome)
Affinity_loc_x = GetLocationVNF(Affinity[0])
Affinity_loc_y = GetLocationVNF(Affinity[1])
Affinity_loc_x = Affinity_loc_y
for each vLink in Chromosome do
    if GetLeftLocation(vLink) == Affinity[0]
    or GetLeftLocation(vLink) == Affinity[1] then
        SetSameLocation(Affinity_loc_x)
    end if
    if GetRightLocation(vLink) == Affinity[0]
    or GetRightLocation(vLink) == Affinity[1] then
        SetSameLocation(Affinity_loc_x)
    end if
end for
Antiaffinity = GetAntiAffinity(Chromosome)
Anti_Affinity_loc_x = GetLocationVNF(Anti_Affinity[0])
Anti_Affinity_loc_y = GetLocationVNF(Anti_Affinity[1])
for each vLink in Chromosome do
    if GetLeftLocation(vLink) == Anti_Affinity[1] then
        SetSameLocation(Anti_Affinity_loc_y)
    end if
    if GetRightLocation(vLink) == Anti_Affinity[1] then
        SetSameLocation(Anti_Affinity_loc_y)
    end if
    if GetLeftLocation(vLink) == Anti_Affinity[0] then
        SetSameLocation(Anti_Affinity_loc_x)
    end if
    if GetRightLocation(vLink) == Anti_Affinity[0] then
        SetSameLocation(Anti_Affinity_loc_x)
    end if
end for
```

convergence time required for large-scale enterprise network instances, where its complexity is NP-Complete, while FASTSCALE will be evaluated both on small and large-scale enterprise networks. However, FASTSCALE will be compared with the ILP solution only on a small enterprise network to assess its solution quality and effectiveness.

2.7.1 Complexity Analysis

The complexity of our cultural genetic algorithm is straightforward, and depends on several parameters and components. For greater clarification, we describe the components individually rather than globally.

We start with NSGA-II Deb, Pratap, Agarwal & Meyarivan (2002), which forms the core of our cultural algorithm. Its complexity is known to be $O(MxN^2)$. Here, N is the population size and M is the number of cost functions within the model. So far, in our mathematical model, we have four objective functions, i.e., $M = 4$. Thus, the complexity of NSGA-II $\approxeq O(N^2)$ since M is just a fixed parameter, and not a variable.

The complexity of our repairing algorithm, GenoFix, is $O(C^2 \times E \times Log(V))$ as it uses Dijkstra algorithm with priority queues, where C is the size of the chromosome, E is the set of physical links and V the set of servers of the network infrastructure. The affinity and anti-affinity algorithm, within the GenoFix algorithm, has a complexity of as it mainly depends on the size of the chromosome. Finally, the complexity of history knowledge is , using the Timsort algorithm to sort the servers in order to reduce the search space of NSGA-II. Thus, the overall complexity of FASTSCALE is $O([V + C^2 \times E \times Log(V)] \times N^2)$. It is worth mentioning that the complexity analysis we have performed is in its worst-case scenario form.

2.7.2 Setup

We implemented our Integer Linear Program using Gurobi 7.51, and our cultural genetic algorithm with Python 2.7 and Platypus framework David Hadka (2015). All the implementations and experiments were run on a physical machine with 8 × Intel® Core™ i7-6700 CPU cores @ 3.40GHz and 15.6 GB of memory. We considered two types of network infrastructures: small enterprise network composed of 20 servers and a Data Center enterprise network with 200 servers. Both the network infrastructure and virtual network service topologies were generated using the NetworkX library. It is worth mentioning that the topology of virtual network services (together with in-line services) forms a complex topology, and is not limited to linear topologies

like most of the proposed solutions in the literature. All the experiments were repeated 10 times and we report the average values.

Tables 2.3, 2.4, 2.5 and 2.6 respectively summarize the parameters of our metaheuristic, the characteristics of the network infrastructure, the characteristics of the virtual network services with the QoS parameter, and the parameters of the mathematical model.

Table 2.3 Parameters of the metaheuristic

Parameters	**Values**
Population size	{30,60,90}
Number of iterations	{30,60,90,150,200,250}
Crossover probability	0.8
Mutation probability	0.25
Normative knowledge	Top %20 of individuals found during the current iteration
Situational knowledge	Best individuals found during the whole evolutionary process

Table 2.4 Characteristics of the network infrastructures

Networks with their characteristics	**Small entreprise network**	**Data center network**
Number of servers	20	200
CPU cores	32	32
Memory (Gig)	1000	1000
Storage (Gig)	1000	1000
Delays (ms)	Set arbitrarily from [30, 60]	Set arbitrarily from [30, 60]
Distance between each pair of servers (hops)	Set arbitrarily from [1, 5]	Set arbitrarily from [1, 5]
Physical link capacities	10 Gbps	20 Gbps

Table 2.5 Characteristics of the virtual network service

Virtual network service size	**Resource requirements (CPU, Memory and Storage)**	**End-to-end delays for virtual links**	**Bandwidth requirements for virtual links**
Virtual network functions ranging from [5, 25]	4, 4 and 32 unit of resources	Set arbitrarily from [100, 500] ms	Set arbitrarily from [200, 500] Mbps

Table 2.6 Parameters of the mathematical model

Parameters	Values
Γ_x	1.0 (monetary units)
Λ	0.01 (monetary units)
ϵ	0.1 (monetary units)
Θ_p	1, i.e., all the resources are equally important
P_x^{max}, P_x^{idle}	2.735, 0.0805 in Kilowatts

2.7.3 Results Analysis

In this section, we evaluate FASTSCALE both on small and Data Center enterprise networks. We begin by evaluating FASTSCALE with the ILP solution on a small-scale network. We then assess it on a large-scale network.

2.7.3.1 Small entreprise network

In this experiment, we evaluate FASTSCALE with the ILP solution without considering affinity and anti-affinity constraints. The reason is primarily to achieve a fair comparison since they were intended for the metaheuristic CGA.

Table 2.7 reports the average runtime per accepted request of FASTSCALE (i.e. CGA) versus the ILP solution as we vary the virtual network size i.e. the number of virtual links composing it in the range Bari *et al.* (2015), Lee & Zomaya (2012). We can clearly see that our meta-heuristic exhibits the lowest runtime with respect to increasing the number of virtual links compared to the ILP solution which increases linearly. It is worth mentioning that this result is obtained for a small number of iterations i.e. 20 we set for FASTSCALE. In fact, this is due mainly to the polynomial complexity of our approach and the belief space we used to reduce the search space thanks to the history knowledge, as it tracks the residual resources of each server after granting each request. Another observation is related to ILP iterations, returned by Gurobi, which increase linearly from 297 to 4613 as we increase the size of the requests from 5 to 20 then

it goes down as the search space starts getting smaller as fewer servers have enough resources to accept the requests. As an outcome, FASTSCALE is time-efficient and able to find solutions quickly compared to ILP.

Table 2.7 Average runtime per accepted request

NS size	CGA runtime	ILP runtime	ILP iterations	CGA iterations
5	0.03490	0.20892	297	20
6	0.04152	0.27908	337	20
7	0.05595	0.38254	531	20
8	0.06423	0.52970	674	20
9	0.07306	0.67902	781	20
10	0.09055	0.73767	1064	20
11	0.10102	0.81229	1182	20
12	0.11218	0.97108	1525	20
13	0.12941	1.28931	1934	20
14	0.13379	1.28770	2342	20
15	0.14056	1.38807	2553	20
16	0.15213	1.63061	3372	20
17	0.17155	2.23236	3125	20
18	0.18427	2.46179	3493	20
19	0.18877	2.44967	3767	20
20	0.19344	2.43104	4613	20
21	0.20010	2.89489	4533	20
22	0.22122	8.11784	1748	20
23	0.23114	8.83404	1968	20
24	0.24822	9.30294	1840	20
25	0.26436	10.2616	2121	20

Fig. 2.3 plots the average runtime per accepted request, while varying the number of iterations and population size. We can clearly see that FASTSCALE exhibits the lowest runtime (0.05 s) with 30 individuals and 30 iterations than does the ILP solution. However, the runtime increases as we increase the number of iterations and population size. This is straightforward since the complexity of NSGA-II is a function of the population size.

Fig. 2.4 reports the average FASTSCALE traffic cost per accepted request as compared to the ILP solution, while varying the number of iterations and population size. Akin to the energy cost, our metaheuristic shows a result that is very close to the ILP solution.

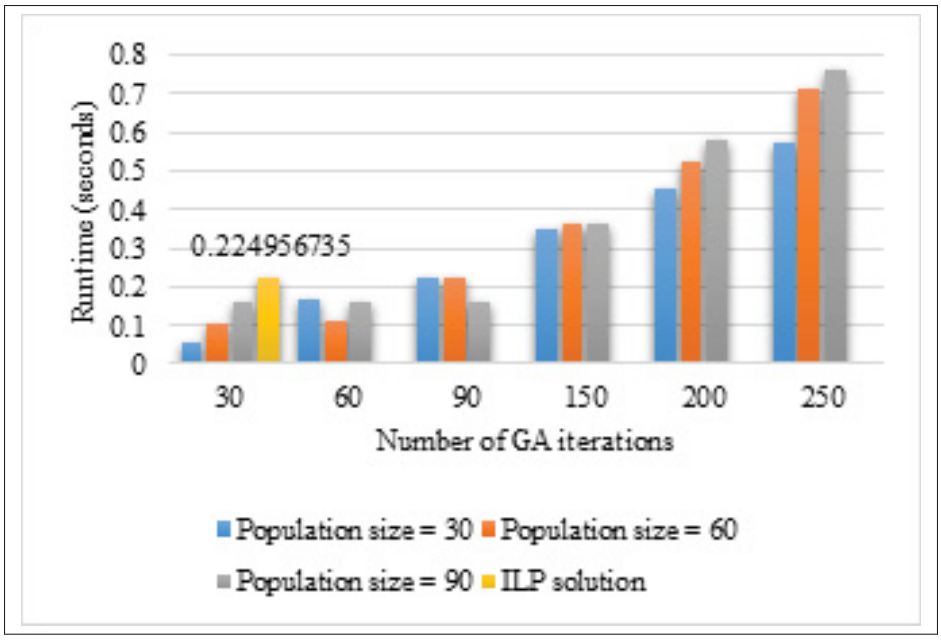


Figure 2.3 Average runtime per accepted request (number of VNFs=5)

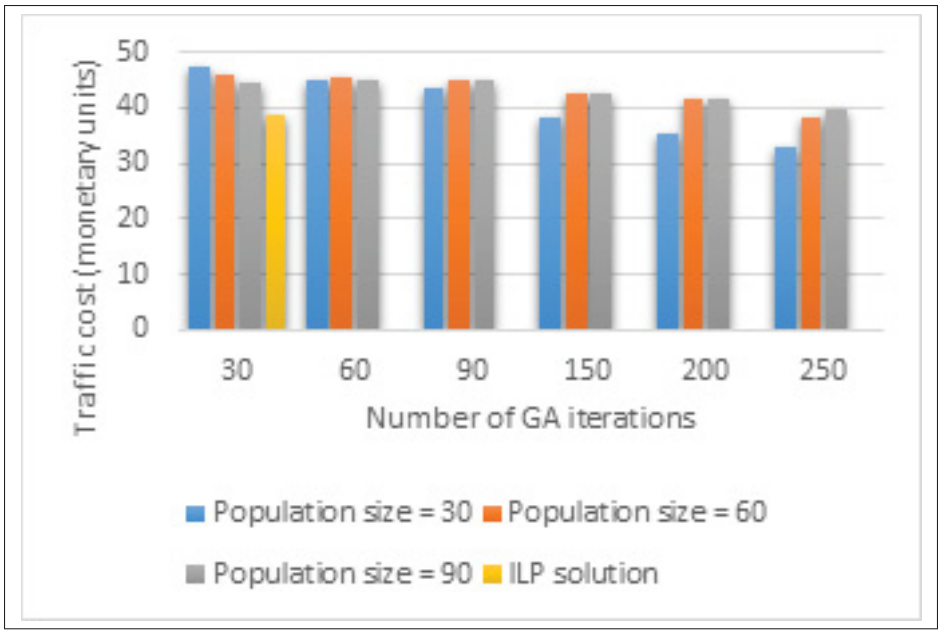


Figure 2.4 Average traffic cost per accepted request (number of VNFs=5)

Fig. 2.5 reports the average FASTSCALE energy cost per accepted request as compared to the ILP solution, while varying the number of iterations and population size. We can clearly see that the ILP solution exhibits the lowest energy cost, as compared to the metaheuristic. It is interesting to see that regardless of the number of iterations and population size, the FASTSCALE energy cost is pretty much the same.

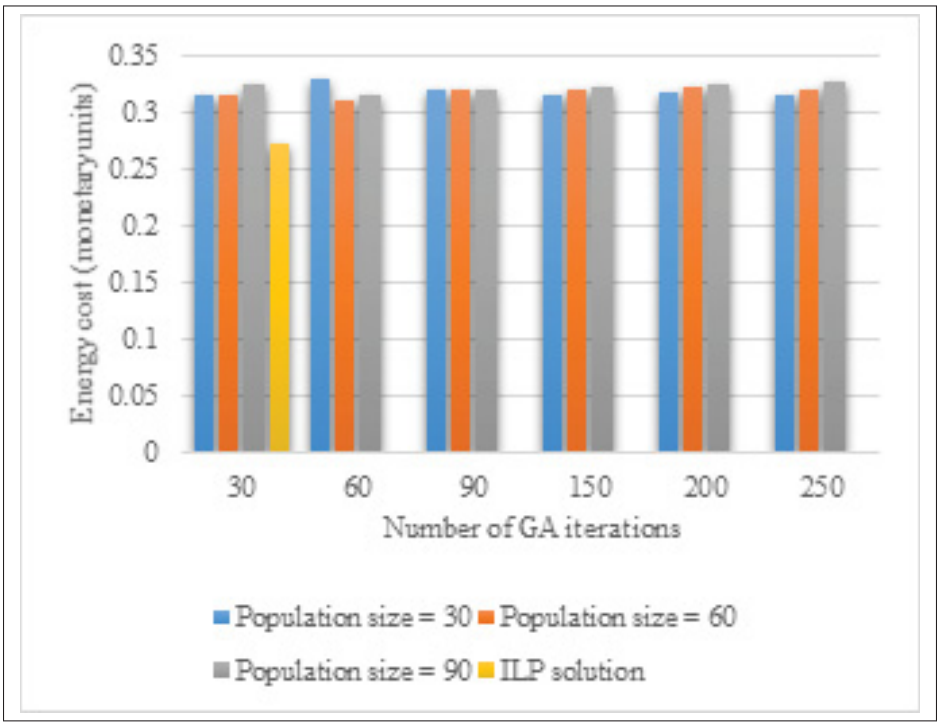


Figure 2.5 Average energy cost per accepted request (number of VNFs=5)

Since FASTSCALE uses the same ILP structure in its core, Figs. 2.3, 2.4 and 2.5 confirm that our approach is efficient and very competitive, although the two approaches ILP and CGA are different. The aim of the following experiments is to answer the question: *How good is FASTSCALE, compared to ILP*? Figs. 2.6, 2.7, 2.8 respectively report the ratio of runtime, traffic cost and energy cost between FASTSCALE and ILP. The main takeaway from Fig. 2.6 is that the lower the ratio, the faster FASTSCALE is, compared to ILP. For instance, FASTSCALE is faster than ILP, regardless of the number of individuals in the range of [30, 60, 90] iterations. Similarly, Fig. 2.7 reports the traffic cost ratio, which shows that ILP is better than FASTSCALE

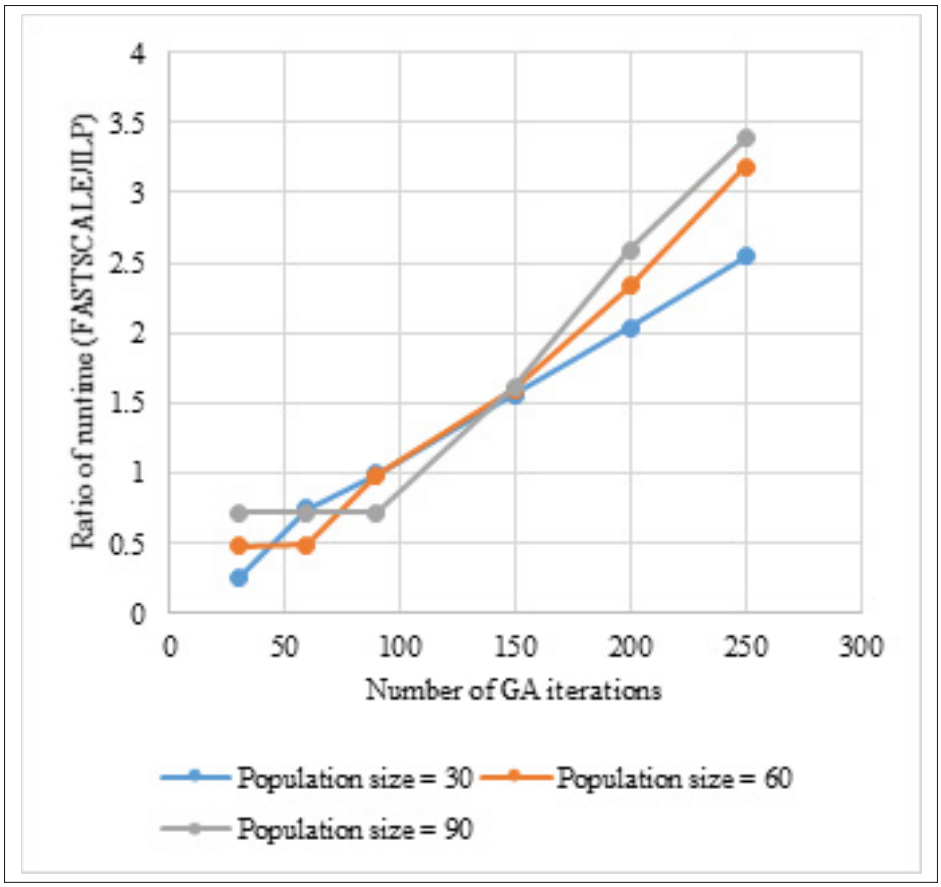


Figure 2.6 Ratio of runtime (number of VNFs=5)

since it tends to pack the VNFs within the same host regardless of the number of iterations with 60 and 90 individuals. However, this is not always desired in real deployments since affinity and anti-affinity may be present, or for performance and security purposes.

Regarding the energy cost, the ILP solution is better than FASTSCALE since ILP performs a more exhaustive search than does FASTSCALE. However, FASTSCALE is competitive since the gap is very small, as compared to the ILP solution.

Figs. 2.9, 2.10 and 2.11 respectively report the average runtime, traffic cost and energy cost for virtual networks composed of 10 VNFs. As we can see from Fig. 2.9, FASTSCALE shows a lower runtime with a population of 30, 60 and 90 individuals for 30, 60 and 90 iterations than the ILP solution. Then, beyond 150 iterations, the ILP solution exhibits a lower runtime

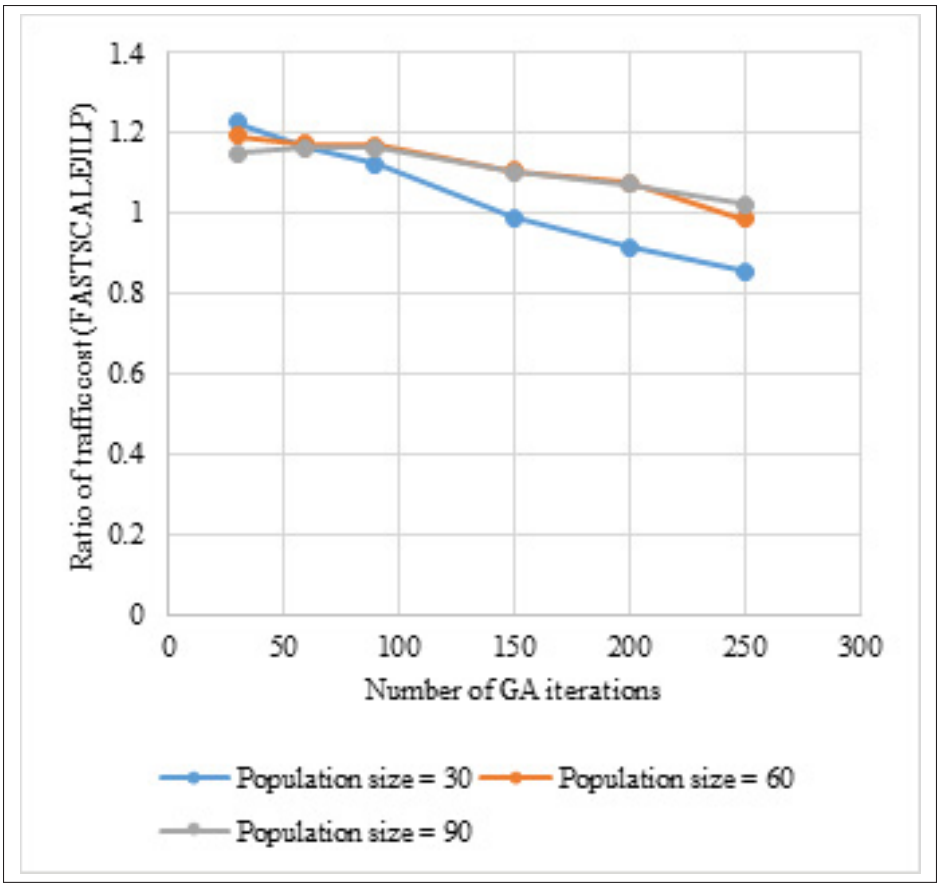


Figure 2.7 Ratio of traffic cost (number of VNFs=5)

compared to FASTSCALE. It is worth mentioning that the ILP bar is shown on the left for comparison purposes only, and is not associated with any iterations like FASTSCALE. From Fig. 2.10, we can clearly see that FASTSCALE exhibits a lower traffic cost as than does the ILP solution regardless of the number of iterations and population size. The ILP solution has lower energy costs than FASTSCALE, as shown in Fig. 2.11, regardless of the number of iterations and population size. This is thanks to the exhaustive search performed by the simplex algorithm, which minimizes the fourth objective function. However, FASTSCALE is still very competitive since the gap is around 0.25 monetary units, as compared to the ILP solution.

Akin to Figs. 2.6, 2.7, 2.8, Figs. 2.9, 2.10 and 2.11 respectively report on how FASTSCALE compares to ILP in terms of runtime, traffic cost and energy cost when a virtual network size is composed of 10 VNFs. From Fig. 2.12, we can clearly see that FASTSCALE is faster than ILP

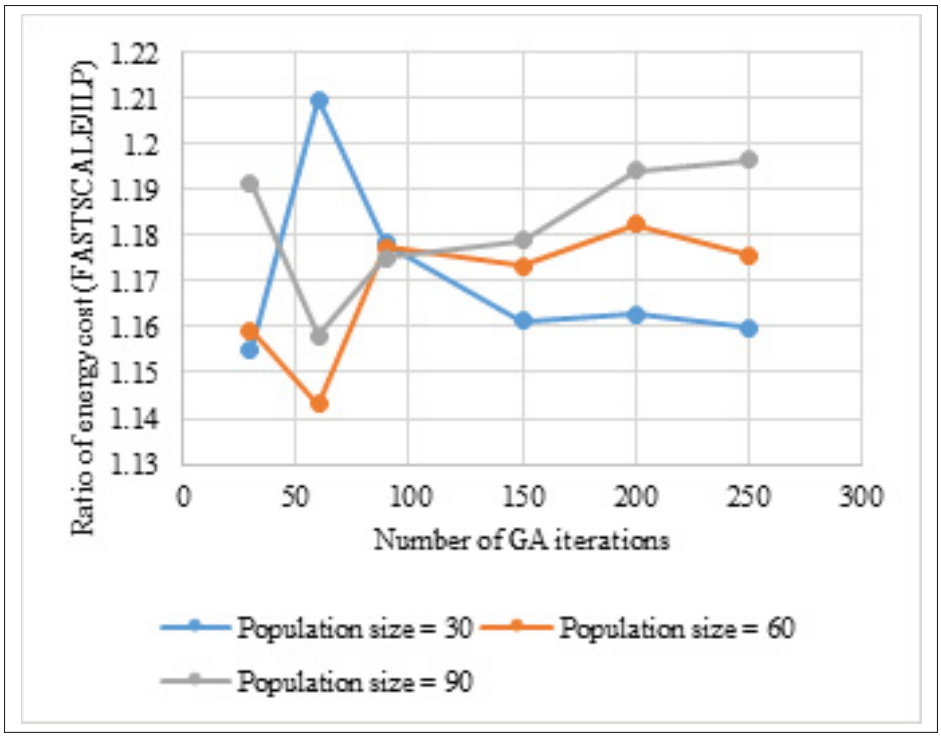


Figure 2.8 Ratio of energy cost (number of VNFs=5)

for a number of iterations less than 150, regardless of the population size. Fig. 2.13 shows that FASTSCALE is better than ILP in terms of traffic cost, regardless of the number of iterations and population size since the ratio is lower than 1.0. However, in Fig. 2.14, the ILP solution is better than FASTSCALE in terms of energy cost since it explores the search space efficiently and exhaustively, thereby minimizing the fourth objective function.

2.7.3.2 Data center entreprise network

In this experiment, we assess our FASTSCALE on a Data Center enterprise network topology. As well, we include and anti-affinity for the virtual network service requests. Moreover, we provide results for a virtual network service of size 10 VNFs.

Figs. 2.15, 2.16 and 2.17 report the average of runtime, energy and traffic costs of FASTSCALE, respectively. From Fig.2.16, FASTSCALE reveals that the number of iterations did not significantly reduce the energy cost since the cost is within the 0.911–0.93 range regardless of

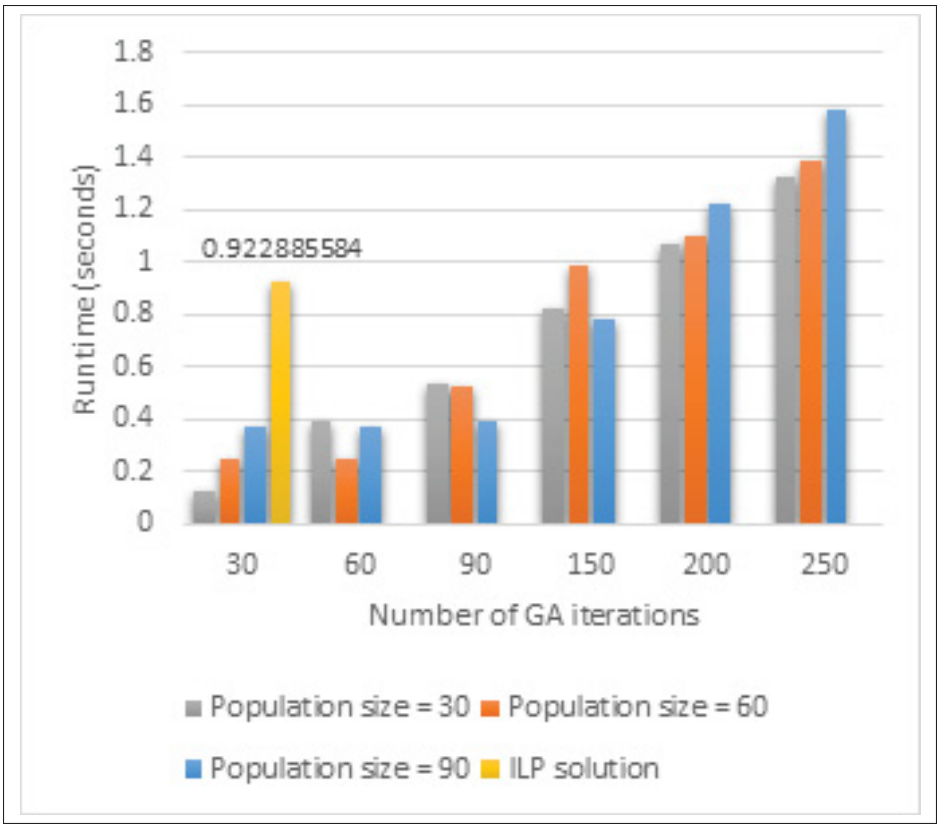


Figure 2.9 Average runtime per accepted request (number of VNFs=10)

the number of iterations and population size. We observe a similar pattern for the traffic cost, as shown in Fig. 2.17, for a population of 60 and 90 individuals, regardless of the number of iterations. However, the traffic cost is reduced by 8 monetary units from 205.64 to 197.05, but to the detriment of the average runtime, as revealed by Fig. 2.15. The reason why the traffic cost does not exhibit a significant gap while varying the population size and the number of iterations is probably linked to the topological structure (diameter less than 6) of the network and the fact that the VNFs are positioned close to each other, since FASTSCALE finds a tradeoff by minimizing the first and second objective functions, respectively.

Fig. 2.18 reports the average runtime of FASTSCALE per accepted request respectively for virtual network services composed of 5 and 10 VNFs. We can clearly see that the size of these

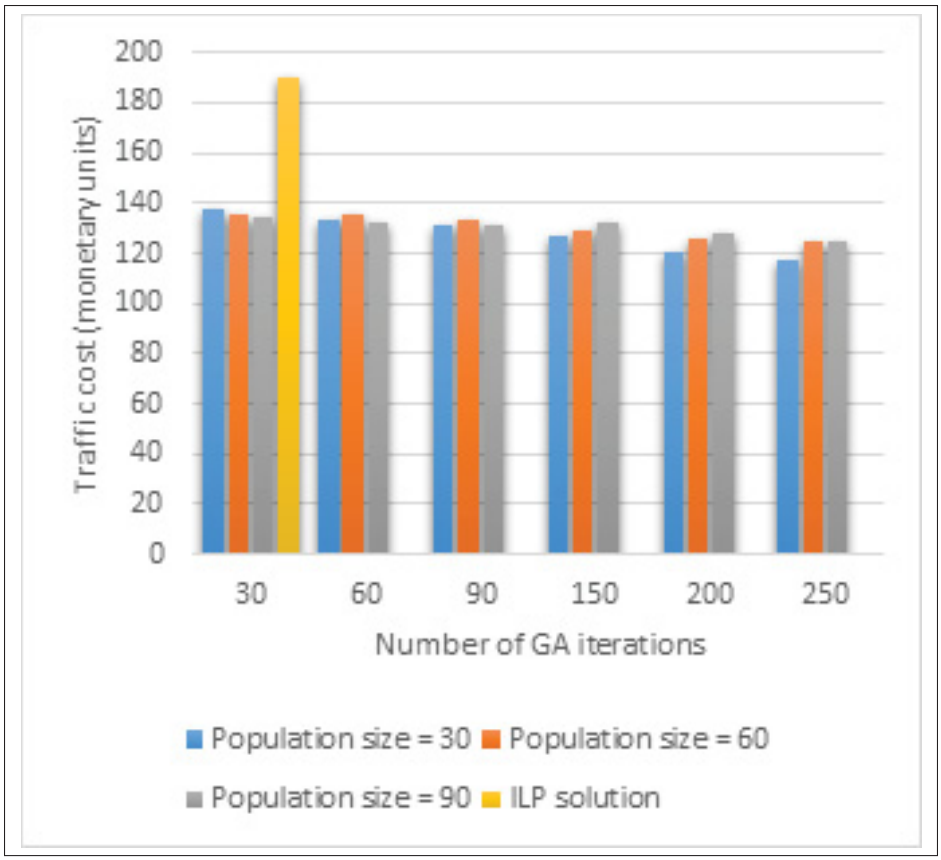


Figure 2.10 Average traffic cost per accepted request (number of VNFs=10)

services, the number of iterations, and the population size have an impact on the runtime since FASTSCALE's algorithmic complexity depends on these factors.

Fig. 2.19 reports on the average number of servers used by FASTSCALE, while varying the number of iterations for a virtual network service request composed of 10 VNFs. We can see that the average number of servers is within 8.2 and 8.8. Yet, for a population of 30 individuals, the average number of used servers decreases as we increase the number of iterations. However, this is to the detriment of the runtime, as can be seen from Fig. 2.18.

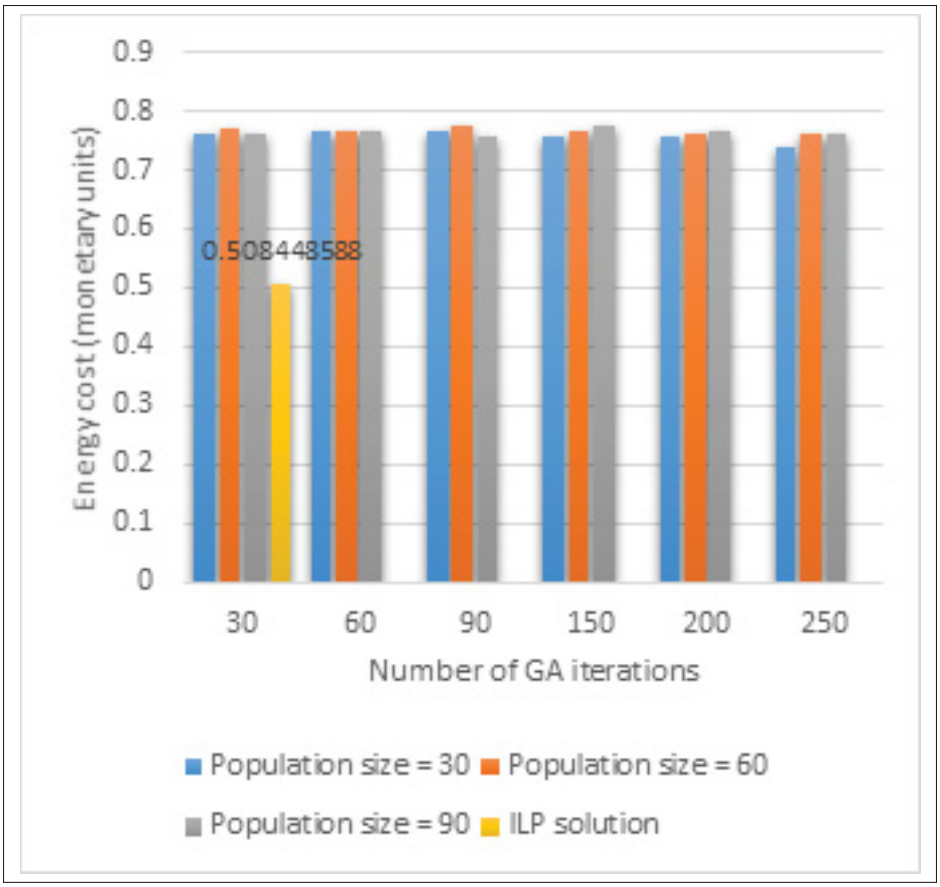


Figure 2.11 Average energy cost per accepted request (number of VNFs=10)

2.7.3.3 Summary

We summarize the main results obtained from our evaluations of the proposed solutions both for small- and large-scale instances of the problem. The main takeaway of this summary is as follows:

1) The quality of results obtained by the proposed metaheuristic are interesting given its competitiveness and effectiveness compared to the exact solution. This is mainly due to the fact that it embodies the same mathematical structure, i.e., in terms of objective functions and constraints, both for a few and several instances of the problem, as shown from the obtained results.

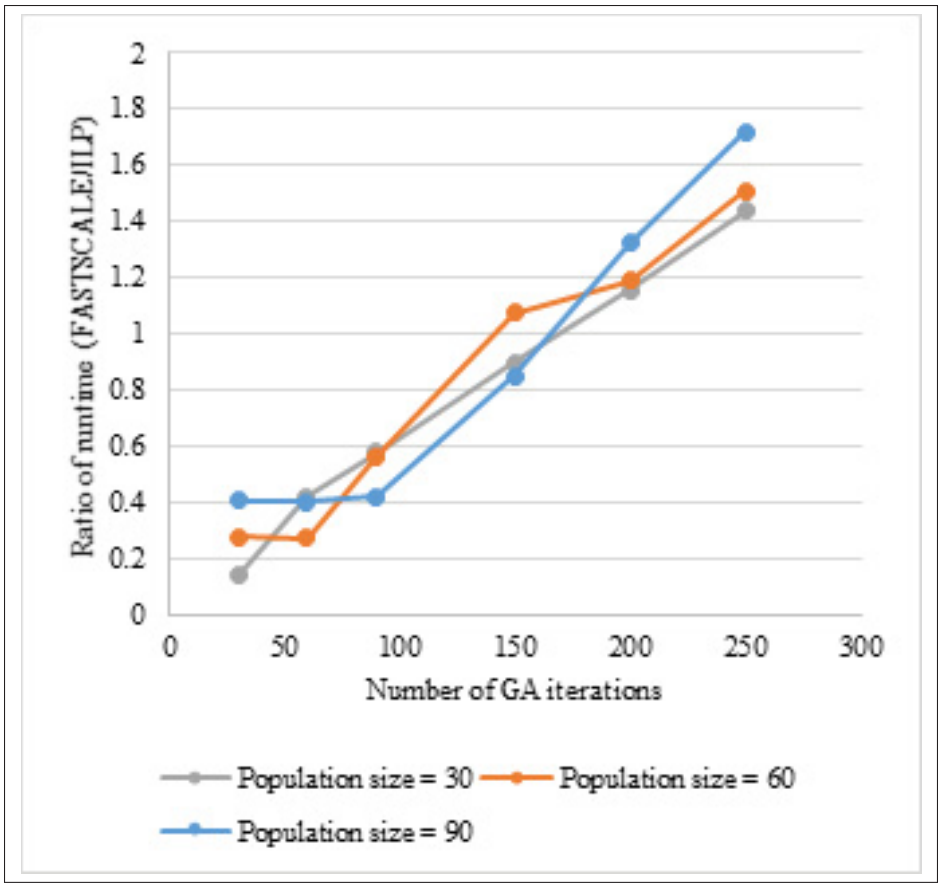


Figure 2.12 Ratio of runtime (number of VNFs=10)

2) For a small number of iterations in a small enterprise network, FASTSCALE exhibits a lower runtime than the exact solution (ILP) for 5 and 10 VNFs, respectively. Moreover, in the case of large instances of the problem, i.e., 10 VNFs evaluated with a data center network, the runtime increases by a factor of 10 as we increase the FASTCALE number of iterations and evolutionary process population size. All the same, the FASTSCALE runtime is still lower than for ILP, which has a longer convergence time for large-scale enterprise networks.

3) The size of the network service requests and network infrastructure affect the average runtime per accepted request, but not the ratio of solutions obtained, as can be seen from the experiments. Clearly, this has to do with the algorithmic complexity of the NSGA-II, as specified in the Complexity analysis section.

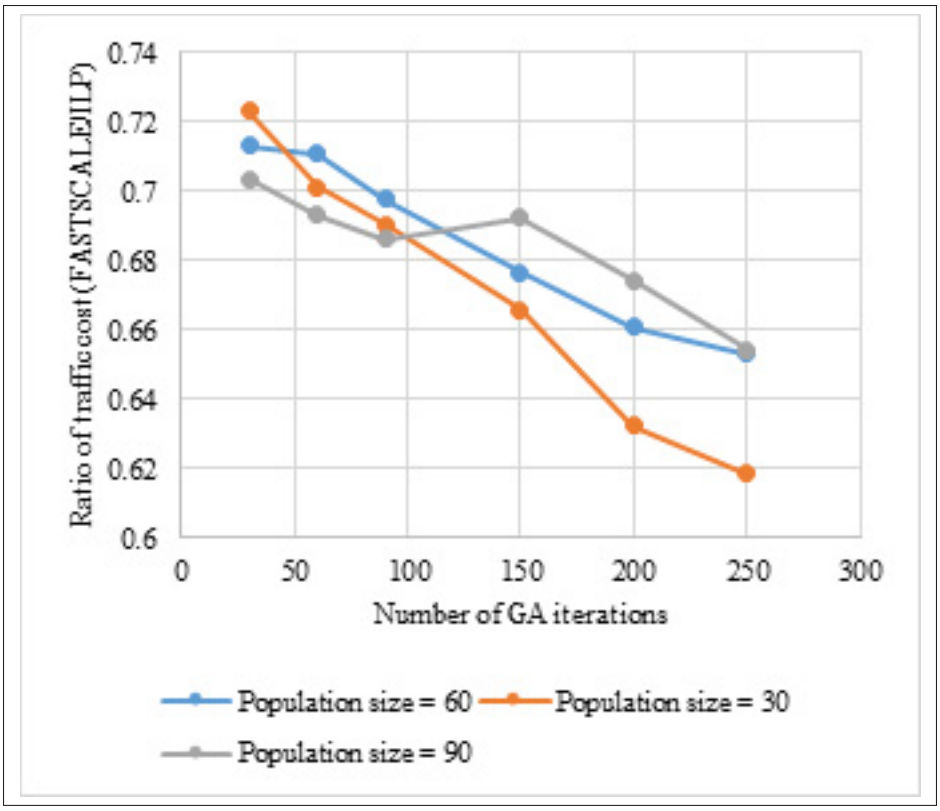


Figure 2.13 Ratio of traffic cost (number of VNFs=10)

4) The average number of servers used is likely to be the same as the number of VNFs that compose a request. This is due to the presence of affinity and anti-affinity rules, respectively. However, their presence does not necessarily have an impact on the ratio as the search space has to be explored with their presence in mind. The quality of these solutions is effectively a function of a tradeoff between the different objective functions with respect to the constraints.

2.8 Conclusion

With the advent of 5G and network slicing, network services are becoming increasingly complex, with more stringent requirements being the new norm. In this paper, we examined the problem of joint placement and chaining of a virtual network service not limited to linear topologies and four VNFs. We formulated the problem as a multi-objective integer linear program with

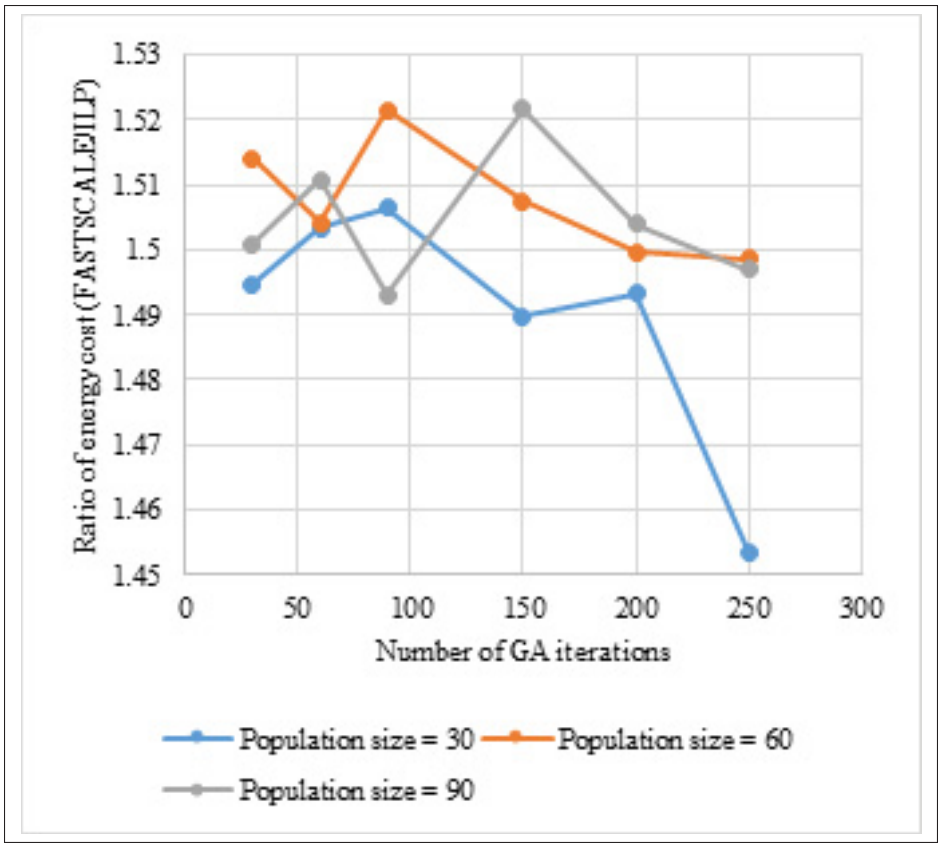


Figure 2.14 Ratio of energy cost (number of VNFs=10)

four conflicting objective functions to be solved optimally. Then, to handle the computational complexity of large-scale instances of the problem, we designed and implemented a fast and scalable evolutionary algorithm, also known as FASTSCALE. FASTSCALE uses the same mathematical structure as the integer linear program. The evaluation and experimental results show that our FASTSCALE approach is faster than the ILP, is scalable, i.e., not limited to three or four VNFs, and provides close to optimal solutions, and remains efficient with fewer iterations and population size variations. The results we obtained so far encourage us to continue exploring certain research avenues within this domain in order to tackle questions such as how to handle traffic fluctuations, incorporate scalability mechanisms and deploy a virtual network service in a multi-domain environment, with the interaction of different actors and SLAs. Probably, one way to reduce the runtime for large-scale instances is to rewrite the implementations from scratch. This can be done from the perspective of low-level and compiled languages such as C, rather

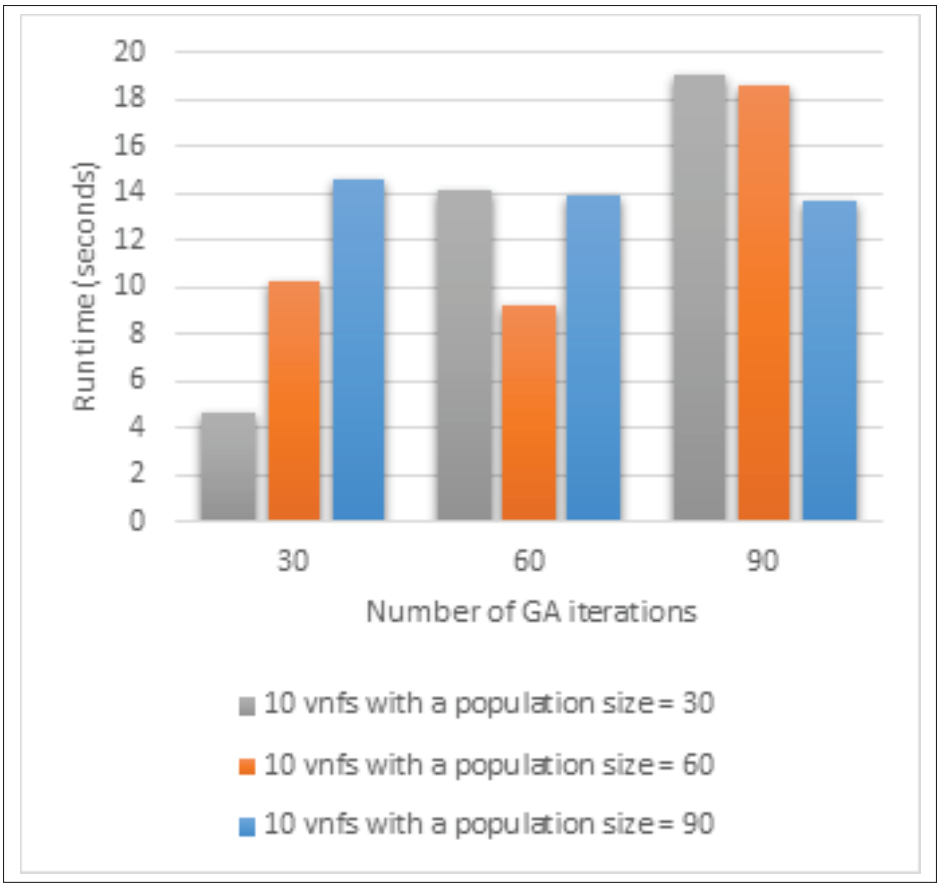


Figure 2.15 Average runtime of FASTSCALE per accepted request (number of VNFs=10)

than that of an interpreted language such as Python, notwithstanding the fact that the complexity of the proposed solutions is polynomial, as shown in Complexity analysis section. Moreover, it is interesting to investigate how to apply a distributed algorithmic approach rather than a centralized one as proposed in this paper by decomposing the problem for large instances into sub-problems with small sizes. In the short term, we plan to investigate the potential extensions related to the multi-domain case both from an Integer Linear Program perspective as well as a metaheuristic perspective. This research avenue may need to reconsider the way the problem is tackled. A distributed solution may likely be designed following the same structure as the exact solution (ILP) and the metaheuristic. For the medium and long terms, we plan to address how to react to traffic fluctuations in addition to scalability mechanisms and how Software

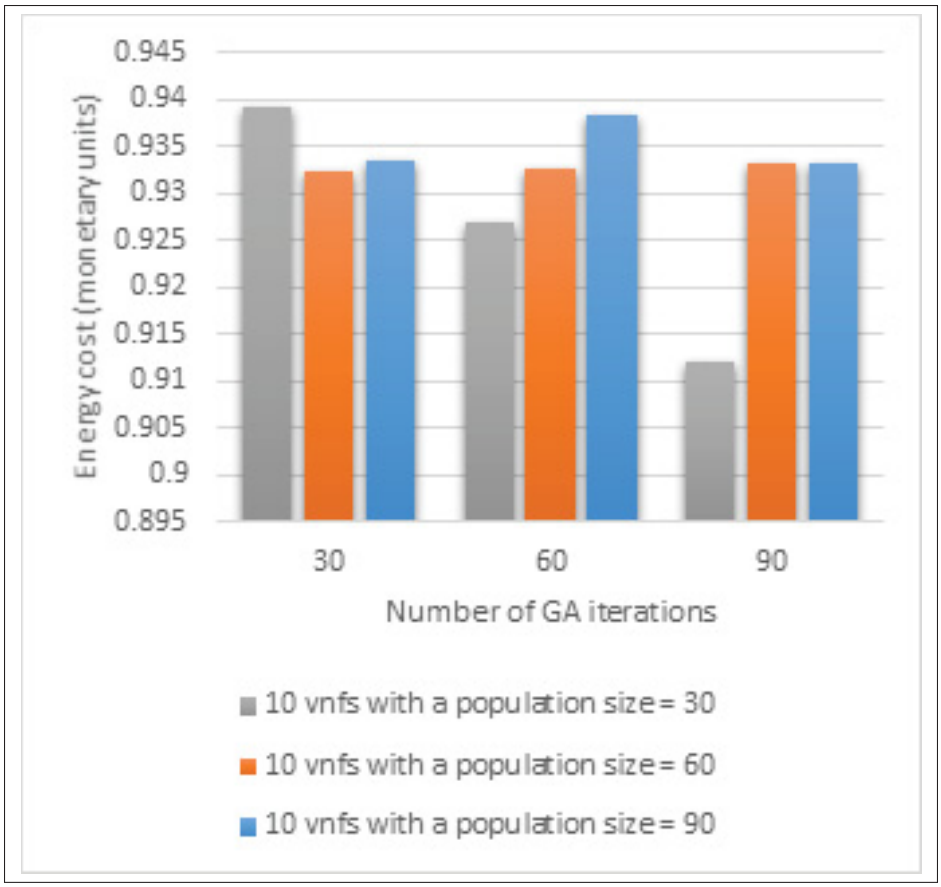


Figure 2.16 Average energy cost of FASTSCALE per accepted request (number of VNFs=10)

Defined Network paradigms could be leveraged. Finally, we plan to develop a platform to test our solutions by investigating existing tools such as OpenStack and Kubernetes according to the MANO framework ETSI.

Acknowledgments

This work has been supported by Ericsson Canada and the Natural Sciences and Engineering Research Council of Canada (NSERC).

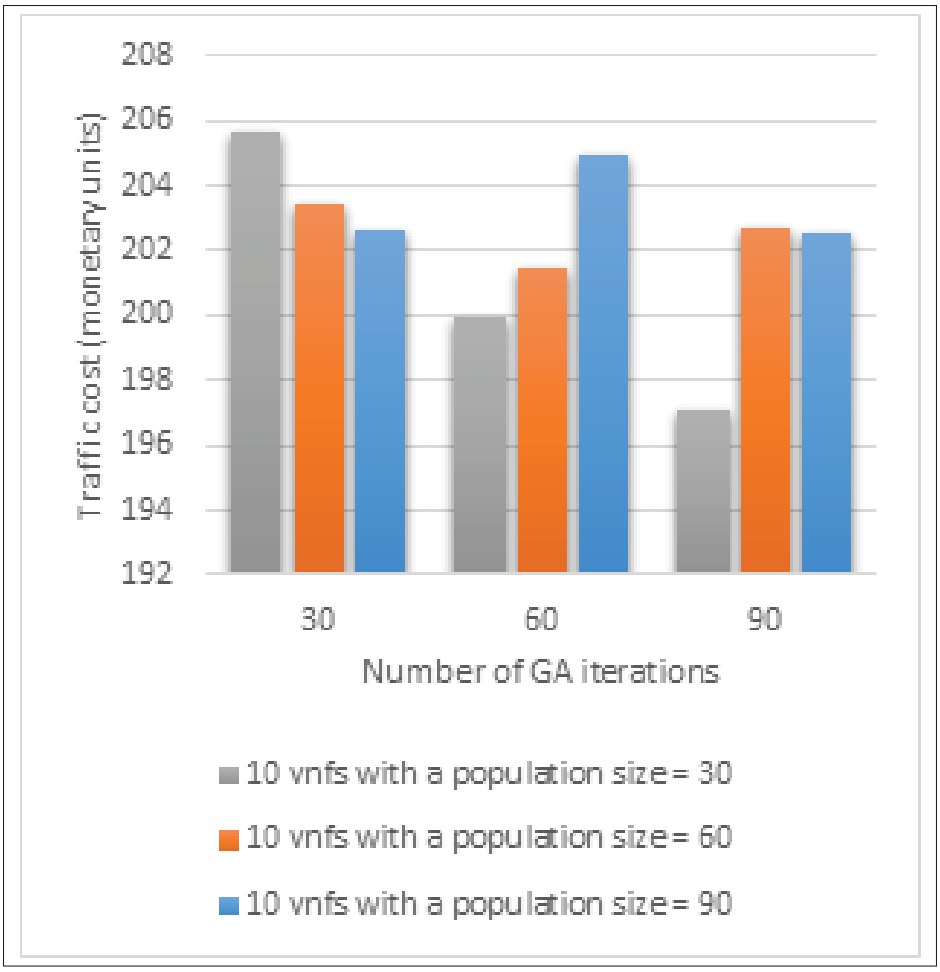


Figure 2.17 Average traffic cost of FASTSCALE per accepted request (number of VNFs=10)

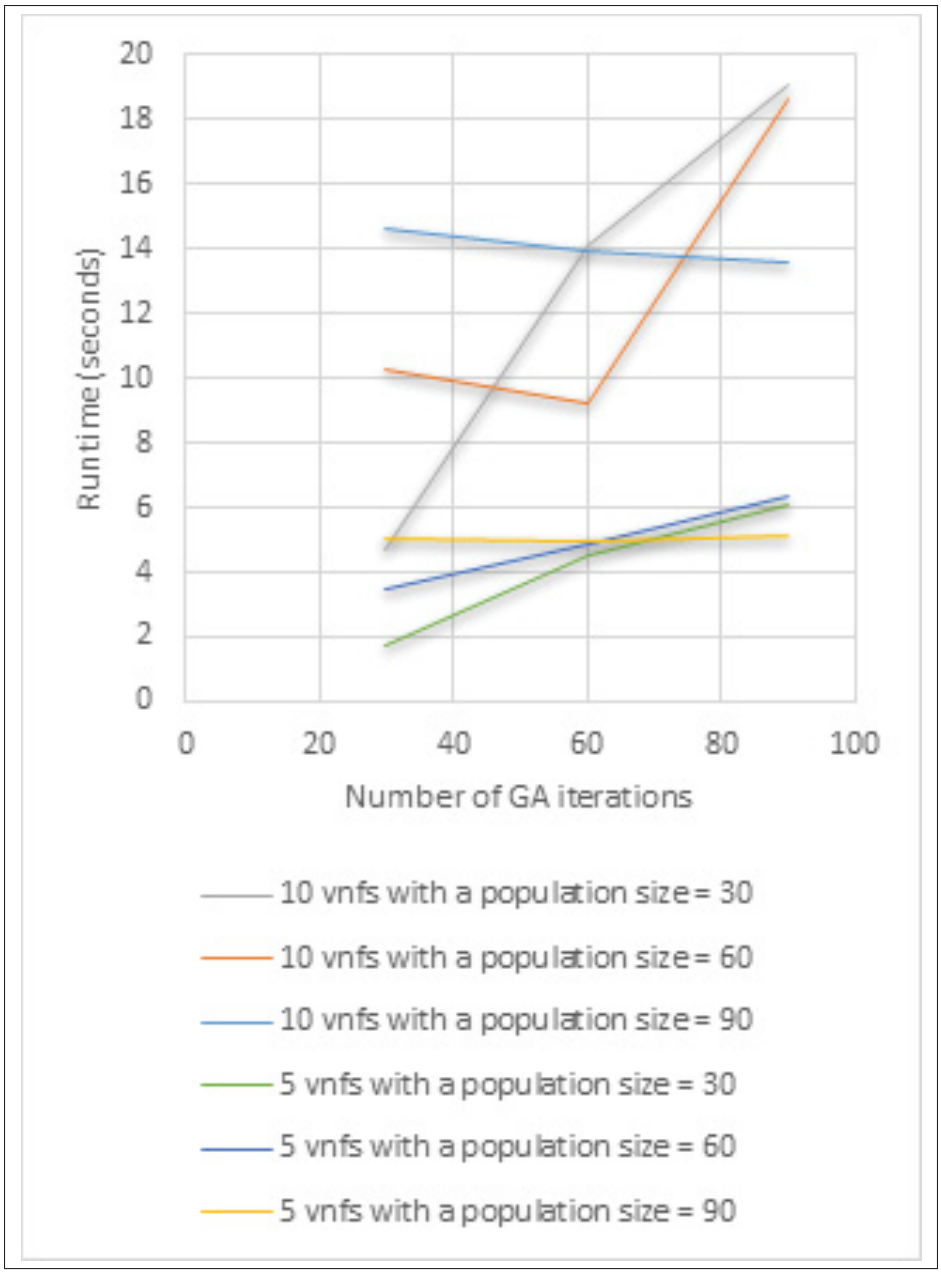


Figure 2.18 Average runtime of FASTSCALE per accepted request

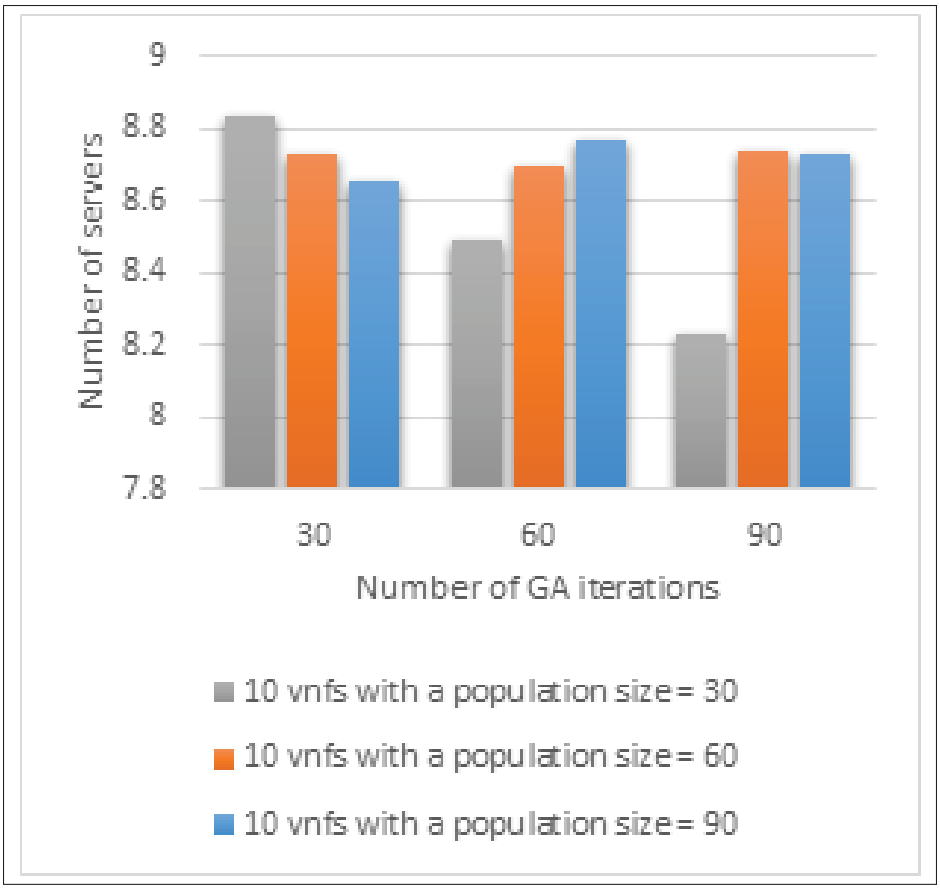


Figure 2.19 Average number of used servers per accepted request

CHAPTER 3

ARTIMIS : A CHEMICAL REACTION OPTIMIZATION APPROACH FOR DELAY-SENSITIVE VIRTUAL NETWORK SERVICES

Laaziz Lahlou[1] , Imane El Mensoum[1] , Nadjia Kara[1] , Claes Edstrom[2]

[1] Département de Génie Mécanique, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3

[2] Ericsson, Canada,
8275 Rte Transcanadienne, Saint-Laurent, QC H4S 0B6

Paper submitted to IEEE Transactions On Cloud Computing on June 2020

3.1 Contributions

This is a joint research work in collaboration with Imane El Mansoum, a graduated master student, who developed the CRO-based approach described in section 3.10.

3.2 Abstract

Service function chaining is a prominent concept that helps network and service providers deploy a set of VNFs according to a policy to realize a network use-case scenario on the fly. The performance of delay-sensitive applications depend on the KPIs of the selected and instantiated VNFs as well as the method used to deploy them. In fact, in the context of 5G and network slicing, these applications tend to be complex and delays must be constantly met to avoid SLO violations. Hence, their deployment require an approach that considers the operational cost, QoS delay constraints and the selection of the appropriate VNFs with the required KPIs jointly. Therefore, two stages need to be addressed in provisioning these types of applications efficiently namely, the selection of the appropriate VNFs and their deployment. The former selects the appropriate VNFs with respect to the required KPIs (e.g., delay), while the latter enforces it to mitigate SLO violations by deploying them effectively. Thus, we propose ARTIMIS, a multi-stage strategy that first selects the appropriate VNFs and then deploys them in an effective

way. Extensive experiments demonstrate the effectiveness and efficiency of ARTIMIS against an existing state-of-the-art approach. Yet, ARTIMIS enables enforcing SLO.

Keywords : Service Function Chain, Chemical Reaction Optimization, Metaheuristic, Resource Allocation, VNF Selection.

3.3 Introduction

Network Function Virtualization has become a key technology allowing IT actors such as network, service and cloud providers to offer telco services and commercial off-the-shelf products on the fly, while leveraging cloud computing capabilities in terms of elasticity mechanisms and enjoying the pay-as-you-go business model.

This promising paradigm helps these actors instantiate a set of VNFs (Virtualized Network Functions) dynamically to build and deploy virtual network services over their infrastructures. A virtual network service consists of a set of virtualized network functions that make up a service function chain Laaziz, Kara, Rabipour, Edstrom & Lemieux (2019). For example, a virtualized IMS typically comprises more than three virtual network functions Laaziz *et al.* (2019). In addition to specific virtualized network functions, attributes are defined for each service function chain. These attributes allow service/cloud providers to express policies that should be satisfied by the virtualized network service. For more information and further reading, we refer the reader to these comprehensive references Mijumbi *et al.* (2016), Yi *et al.* (2018).

Delay-sensitive virtual network services such as mission- critical services and conversational services, to name a few, have stringent requirements in terms of end-to-end delay Ye, Zhuang, Li & Rao (2019). Indeed, it is among the most relevant QoS parameters that drive the design of efficient placement and chaining mechanisms, along with dynamic adaption of resources and VNFs in virtualized environments, which help cloud and service providers meet clients' SLOs (Service Level Objectives) and avoid its violation to prevent penalty costs and overall performance degradations. Another prevalent aspect is that most of the optimization models and proposed approaches do not consider the typical characteristics of VNFs and assume that they

are similar in terms of resource consumption and configurations, although different types and VNF vendors or providers are involved DataSwitchWorks, Juniper Networks. Of course, this is not the case. Service providers and VNF vendors offer VNFs with different characteristics and configurations. Thus, the same VNF of a given type with different resource consumptions may provide different performance levels, (e.g., latency, throughput, etc.), depending on the circumstances DataSwitchWorks, Juniper Networks.

In fact, there is a hidden interplay between the performance of the virtual network service and the achieved individual performances of the VNFs and the intended use case. It is crucial to carefully select the VNFs, with respect to their types and the targeted service goals to avoid performance degradations, unpredictable processing times that may impact the end-to-end delay, service availability, etc. Nevertheless, service and/or cloud providers must consider both the dimensioning of virtual network services, i.e., the selection of appropriate VNFs according to the desired KPIs, and the mechanism that will help deploy them while meeting the SLA as agreed upon with clients.

Deploying service function chains, or virtual network services in general, involves the selection of the VNFs to instantiate, as well as placing and chaining them. However, an effective and efficient deployment requires an efficient selection of these VNFs, along with an equally efficient placement and chaining mechanism that helps deploy them effectively with respect to the goals targeted by the service/cloud provider, while fulfilling the SLO agreed upon on with clients. Therefore, and to the best of our knowledge, it is important to highlight that neither the mathematical formulation presented nor the meta-heuristic approach elaborated for delay-sensitive virtual network services, were addressed in the literature D'Oro, Palazzo & Schembra (2017b), D'Oro, Galluccio, Palazzo & Schembra (2017), Wang, Lan, Zhang, Hu & Chen (2015), Laaziz *et al.* (2019), Pham *et al.* (2020), Khebbache *et al.* (2018), Khebbache *et al.* (2017), Pei, Hong, Xue & Li (2019), Draxler, Karl & Mann (2018), Fischer, Bhamare & Kassler (2019), Gouareb, Friderikos & Aghvami (2018). Moreover, the literature does not address, explicitly the selection of the VNFs prior to their placement and chaining.

Our aim in this paper is to propose a multi-stage approach to tackle the problem of the placement and chaining of virtualized network services, which are composed of several virtualized network functions (VNFs), taking into consideration the selection of VNFs with respect to some Key Performance Indicators (KPIs) aspects such as the resiliency level, response time and/or throughput prior to their deployment.

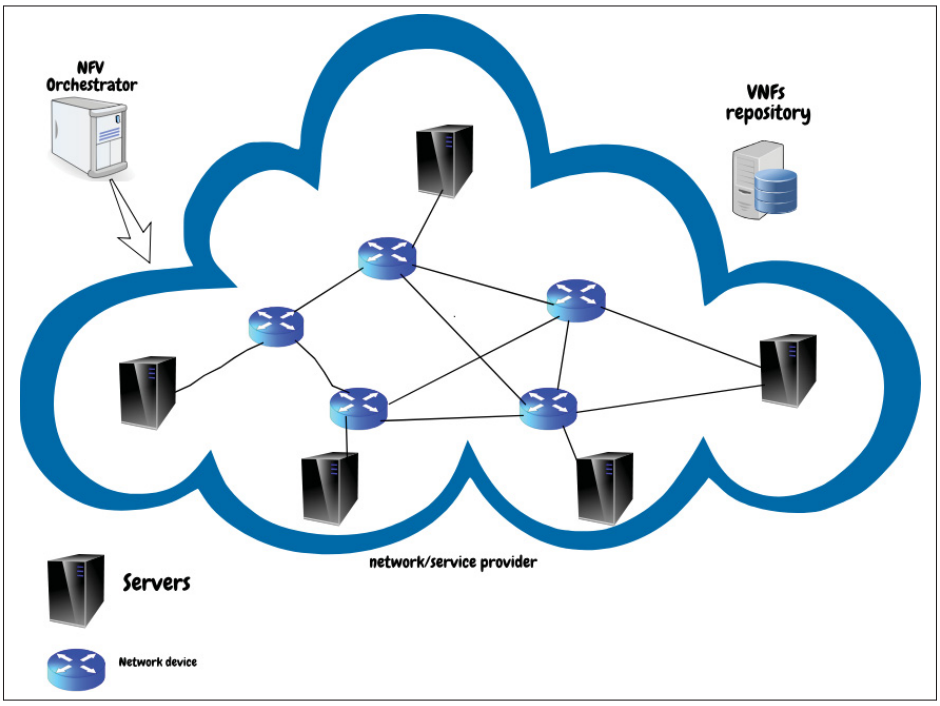


Figure 3.1 An example of a network scenario that embodies a network/service provider with a set of VNF licenses

In this paper, we fill this gap in the literature with the following key/main contributions:

1) We propose a VNF selection model to find the best VNFs that match the requested and expressed Key Performance Indicators (KPIs) and a suite of solutions, namely, an Integer Linear Program, an exhaustive search based-algorithm and a Pareto Dominance-based genetic algorithm;
2) We provide a formulation for the joint placement and chaining of VNFs for virtual network services, not limited to four VNFs and service function chains with linear topologies;

3) We propose ARTIMIS, a fast and scalable CRO (Chemical Reaction Optimization) approach to handle medium- and large-scale instances of the problem, leveraging the same (ILP) Integer Linear Programming structure as in Laaziz *et al.* (2019);
4) The performance of ARTIMIS in terms of solution quality and scalability is assessed using realistic topologies (i.e., small enterprise and data center network topologies);
5) The results demonstrate that ARTIMIS outperforms the-state-of-the-art CGA (Cultural Genetic Algorithm)-based approach Laaziz *et al.* (2019).

The rest of the paper is organized as follows. We present the system architecture adopted in Section 3.4 and review relevant works in Section 3.5. Next, we describe the theoretical foundations and problem statement for the selection and the placement of virtual network services in Sections 3.6 and 3.7. Yet, within Sections 3.8 and 3.9, we present the proposed suite of solutions for the selection of VNFs prior to the deployment of virtual network services in Section 3.10. The asymptotic analysis, of the proposed solutions, is presented in Section 3.11 followed by the performance evaluation in Section 3.12. Finally, Section 3.13 concludes this paper and identifies avenues for future research.

3.4 System Architecture

Figure 3.2 shows the architecture of ARTIMIS we designed for the deployment of virtual network services. It comprises a control plane and a data plane. The selection of VNFs, based on their requested KPIs, is performed thanks to the VNF selection algorithm. Once this step is completed, the CRO-based deployment procedure is invoked to deploy the virtual network service with the previously selected VNFs. This is done in line with the virtual network service which describes how the VNFs are interconnected. These two procedures take place on the control plane. We consider the virtual network service topology to be complex and not necessarily linear, as are most of the existing solutions proposed in the state of the art D'Oro *et al.* (2017b), D'Oro *et al.* (2017), Wang *et al.* (2015), D'Oro, Galluccio, Palazzo & Schembra (2017a), Pham *et al.* (2020), Khebbache *et al.* (2018), Khebbache *et al.* (2017), Pei *et al.* (2019), Mechtri *et al.* (2016), Draxler *et al.* (2018), Fischer *et al.* (2019), Gouareb *et al.* (2018). The CRO-based

procedure leverages information (available CPU cores, memory, bandwidth, etc.) provided by the control plane, which are fetched on a periodic basis, using analytic tools such as Prometheus Prometheus, as well as the current network infrastructure topology (the available nodes with their interconnectivity (infrastructure links)).

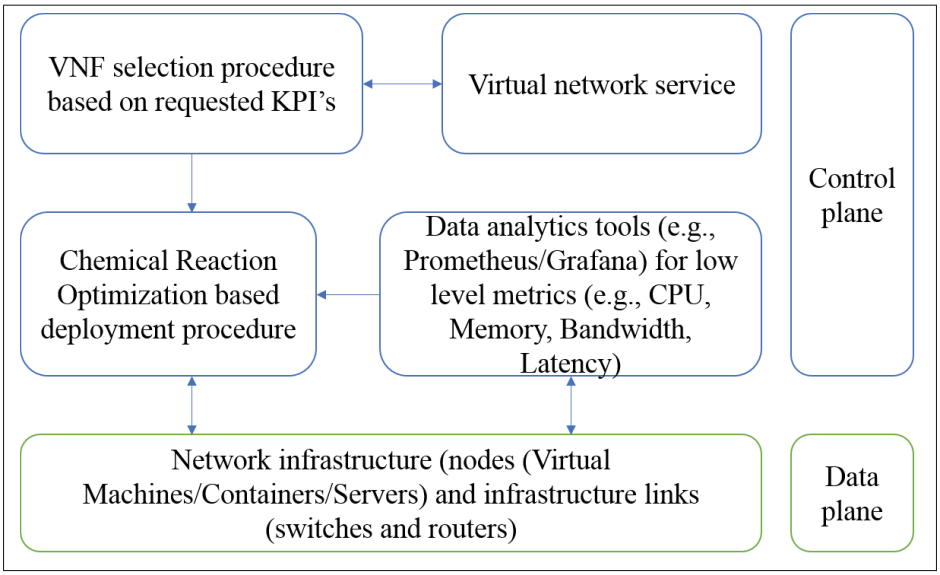


Figure 3.2 The architecture of ARTIMIS for selecting the VNFs composing a virtual network service prior to their deployment

Unless otherwise specified, our system architecture is specifically designed for the network scenario we considered in this paper, as depicted in figure 3.1. We look at a network/service provider who possesses a network infrastructure and a set of VNF licenses that it instantiates to deploy and provision the virtual network service requests it receives from clients.

3.5 Related Works

In this section, we review the existing and most relevant approaches that have thus far been proposed in the research literature according to the selection of VNFs with or without their placement and chaining.

3.5.1 Selection

In D'Oro *et al.* (2017b), the authors propose a marketplace model to understand the interactions between VNF sellers, users and Telco Operator actors in a distributed scenario where each user decides autonomously to compose the service function chains. The users execute a distributed algorithm that helps with the selection of the desired VNFs with respect to a set of requirements such as the latency, reduce the congestion level on the servers hosting the VNF instances, and the price associated with the instantiation of VNFs. Yet, the proposed approach in D'Oro *et al.* (2017b) is an extension of the idea initiated in D'Oro *et al.* (2017) following the same sight line. To help users deploy their services, VNFs are selected according to a pricing model which is analyzed using game theory D'Oro *et al.* (2017a). The proposed approach integrates a set of brokers, which are responsible for selecting and offering users the VNFs fetched from the VNF sellers to satisfy the latter's requirements in terms of QoS and price. These VNFs are offered in the form of VNFaaS (VNF as a Service) following a cloud computing principle.

Compared to D'Oro *et al.* (2017b), ARTIMIS assumes that the Telco Operator has a set of available VNFs to offer to end users, and helps them select the VNFs that satisfy their requirements in terms of QoS and price. Yet, this is done in a centralized fashion, where the Telco Operator focuses on its own profits, and does not compete with other actors or rely on an external entity such as a broker.

In Wang *et al.* (2015), the authors formulate the dynamic composition of VNFs for service function chains as a multi-dimensional knapsack problem and propose a distributed algorithm based on Markov approximation to handle a large instance of the problem. The proposed model and approach parse the requirements of a set of applications into a sequence of service function chains where the VNFs are shared between the applications. Then, the selection of VNFs is performed based on the requirements in terms of different service capabilities and costs. However, in order to enhance the flexibility and the adaptability of the virtual network service, a procedure must be used allowing the network/service provider to deploy the VNFs and readjust the resources and their locations within the underlying networking topology. This is not

achieved only with the service composition mechanism but calls for a placement and chaining algorithm as well.

Compared to Wang *et al.* (2015), we offer these two mechanisms, which are complementary. The former selects the VNFs according to their requirements, whereas the latter attempts to deploy them while fulfilling the requirements with respect to the state of the networking topology in terms of available resources. Yet, we do not share the VNFs between virtual network services as this hinders the identification of performance issues amongst the virtual network services. Moreover, dynamically changing the VNFs at runtime may not be a good option as that may affect the overall performance of the virtual network service, causing service disruptions. Rather, the appropriate route would be to look at the entire virtual network service, and to then opt for scalability mechanisms without changing the requirements of the SLA agreement. A careful examination of the mathematical model presented in Wang *et al.* (2015) reveals that the topologies of the virtual network services that are considered are only linear since the model has the same core as a multi-dimensional multi-knapsack problem.

3.5.2 Placement and Chaining

In Pham *et al.* (2020), the authors propose a joint placement and chaining approach to solve the problem using sampling-based Markov approximation, combined with matching theory. To consider both operational and traffic costs (i.e., physical resource consumption and bandwidth/delay costs), the authors propose a weighted summation as an objective function where the weight set for each cost is defined by the provider and can be adjusted to achieve the targeted trade-off. Moreover, the proposed approach is based on a two-stage algorithm where the first step consists in finding the optimal set of servers to host the VNFs, and the second in placing them in such a way as to minimize the total cost, which means that the placement and chaining are performed in a sequential scheme, and not in parallel. However, the authors argue that their approach has a fast convergence time to the optimal solution, but no indication is given with respect to the execution time. Yet, the simulation shows that as the size of the network grows, the algorithm results in more rejected VNFs in each matching iteration. In our

work, we consider the end-to-end delay, part of the optimization model, which is important for delay-sensitive requests.

In Khebbache *et al.* (2018), the authors propose a meta-heuristic based on a genetic algorithm (NSGA-II) to find Pareto optimal fronts that respect the considered metrics and obtain near optimal trade-offs. As objective functions, the article considers both the mapping cost in terms of resource utilization on hosting nodes as well as link resource consumption. To analyze the performance of their proposed approach, the authors compare their results to those previously presented in Khebbache *et al.* (2018) where they formulate the same problem in two approaches. The first one considers an exact mathematical model to solve the case where service function chains with a maximum of 3 VNFs are hosted. The model considers resource consumption in terms of link utilization only. The main idea is to form cycles of the requested VNFs by adding a fictitious arc from the last to the first VNF, and then solving the substrate graph to find cycles that meet the constraints of link resources. This is known as the Perfect 2-Factor problem, and it has a polynomial time complexity as stated in the article. However, this model cannot scale beyond 3 VNFs per service function chain Mechtri *et al.* (2016). Compared to their works, we consider a multi-resource VNF requests with their required end-to-end delays to be met when deploying them over the network infrastructure.

In Pei *et al.* (2019), the authors formulate the VNF chains placement problem as a Binary Integer Programming model (BIP) to minimize the embedding cost, defined as the total of resource and placement costs. The resource cost represents the cost of the remaining resources on links, on nodes and on VNF instances, while the placement cost represents the computing power and network utilization cost, as well as licensing fees. Since this problem is widely known to be NP-hard, the article proposes two algorithms to optimize the VNF placement. The first one is called the SFC embedding Algorithm (SFC-MAP) and is used to place service function chain requests at minimal cost. The second algorithm is called the VNF dynamic release algorithm (VNF-DRA) which runs periodically and calculates the rate of resource usage on each VNF. However, for the service function chain requests, they are all considered to have 3 VNFs and the topology of the resulting request is linear. Compared to their work, we do not share VNFs

between tenants in order to maximize the acceptance rate since sharing VNFs between tenants may deteriorate the performance of the entire services. Yet, we use only one copy of the network infrastructure to avoid excessive computations when computing a feasible mapping solution.

In Draxler *et al.* (2018), the authors proposed a new approach for mapping service function chains on the substrate network. This consists in considering all possible hosts for a given component and computing the overall cost function they would induce. The choice of a hosting server is made based on the node, ensuring the lowest flow cost in finding the paths between components and the link with the highest bandwidth is therefore attributed the highest priority. Moreover, this joint approach considers spiting a connection between two endpoints on two different paths. To evaluate the performance of their approach, the authors solved their MILP program first by using Gurobi as a baseline comparison with the presented heuristic where they considered a CDN (Content Delivery Network) service composed of 4 VNFs. Howerver, the optimality gap jumps to to 60% for large networks which is very high. Compared to their work, we propose a metaheuristic that improves an initial solution throughout several iterations and we do not split the connection between VNFs along different paths.

Authors in Alleg *et al.* (2017) investigated the impact of a flexible resource allocation on the response time of VNF instances. The purpose of this work is to understand how the VNFs perform under a specific resource allocation scheme in terms of required resouces to satisfy the requested latency while avoiding overallocation of network's resources. To this end, the authors assume a linear dependency between the resulting delay and the allocated resources to the VNFs by leveraging the Amdahl's law. However, the authors proposed only mixed integer linear programs without suggesting any near-optimal solution, which makes the solution solvable for offline requests because of its high complexity. In this paper, we jointly propose a mixed integer linear program and a chemical reaction-based meta-heuristic method for offline and online requests.

Authors in Gouareb *et al.* (2018) studied the use of vertical scaling for the purpose of minimizing network latency. The problem is formulated as a Mixed Integer Linear Program (MILP) and

a heuristic is proposed with the help of k-shortest path algorithm. Basically, the proposed approach takes a batch service function chain requests as input and depending on the overall resource requirements of the shared VNFs vertical and horizontal scaling is used to afford their demands. Moreover, a set of shortest paths are precomputed in order to place and chain the VNFs. However, the authors proposed a randomized algorithm that first tries to place the VNFs and if the operation succeeds it chains the VNFs following the k-shortest paths provided as input. Therefore, the algorithm suffers from high rejection rate since the placement and chaining are not performed jointly. Compared to their work, we consider jointly the placement and chaining of the VNFs within the optimization model and we perform the mapping of the requests following our metaheuristic.

A simulated annealing-based delay sensitive service function chain embedding method is proposed in Fischer *et al.* (2019). In order to evaluate the proposed approach, the authors propose a set of problem references and compare their approach against their previous work. However, simulated annealing-based approaches suffer from higher execution time and the difficulty of tuning its different parameters to work for arbitrary service functions. Yet, the suggested method is tied to linear and simple service function chains. Compared to their work, the proposed meta-heuristic consider the placement and chaining of the VNFs jointly when searching for a feasible solution.

Authors in Ye *et al.* (2019) proposed a queueing model to investigate the estimation of the end-to-end delay when flows traverse a group of VNFs composing a linear service function chain. The authors compared their model with a real simulator to assess the effectiveness of the model and the results outline the importance of considering the end-to-end delay when developing service function chain embedding approaches. However, the authors did not suggest any method that would work in tandem with the proposed end-to-end model.

In contrast to these approaches, we jointly perform the placement and chaining of VNFs for complex virtualized network services after the selection procedure. Several research works D'Oro *et al.* (2017b), D'Oro *et al.* (2017), D'Oro *et al.* (2017a) assume that the VNFs are

already deployed and the topology of the formed network service is linear D'Oro *et al.* (2017b), D'Oro *et al.* (2017), Wang *et al.* (2015); D'Oro *et al.* (2017a), Pham *et al.* (2020), Khebbache *et al.* (2018), Khebbache *et al.* (2017), Pei *et al.* (2019), Mechtri *et al.* (2016). The chaining or flows routing is handled by the network service broker after the VNFs that will process the flows from the marketplace are selected D'Oro *et al.* (2017b), D'Oro *et al.* (2017), D'Oro *et al.* (2017a). In comparison to D'Oro *et al.* (2017b), D'Oro *et al.* (2017), Wang *et al.* (2015), D'Oro *et al.* (2017a), Laaziz *et al.* (2019), Pham *et al.* (2020), Khebbache *et al.* (2018), Khebbache *et al.* (2017), Pei *et al.* (2019), Mechtri *et al.* (2016), Draxler *et al.* (2018), Fischer *et al.* (2019), Gouareb *et al.* (2018), Alleg *et al.* (2017), Ye *et al.* (2019), after selecting the VNFs according to the QoS and resource requirements, we deploy the virtual network service using our chemical reaction optimization procedure. Finally, table 3.1 summarizes the above references.

Table 3.1 Our work versus existing papers

References	Placement & chaining of VNFs	Type of requests	VNFs selection strategy	Optimization technique	Comments
D'Oro *et al.* (2017b), D'Oro *et al.* (2017), D'Oro *et al.* (2017a)	No	Linear	VNFs are selected according to a pricing model using Game Theory	Game Theory	A broker is invoked to select the VNFs.
Wang *et al.* (2015)	No	Linear	VNFs are selected according to their KPIs	Markov approximation	VNFs are shared between requests.
Pham *et al.* (2020)	Yes	Linear	Not addressed	MILP + Markov approximation & Matching theory	Higher rejection rate and the objective functions not expressed with the same unit.
Khebbache *et al.* (2018), Khebbache *et al.* (2017)	Yes	Linear	Not addressed	NSGAII, heuristic	Considers only CPU for the VNFs, the delay is not considered and cannot scale beyond 3 VNFs.
Pei *et al.* (2019)	Yes	Linear	Not addressed	MILP + heuristic	VNFs are shared between tenants and the objective functions are not expressed with the same unit
Mechtri *et al.* (2016)	Yes	Arbitrary	Not addressed	MILP + heuristic	Considers only CPU and bandwidth, ignores the delay and the VNFs are shared between tenants.
Draxler *et al.* (2018)	Yes	Arbitrary	Not addressed	MILP + heuristic	The optimality GAP and execution time are higher. It supports path splitting.
Laaziz *et al.* (2019)	Yes	Arbitrary	Not addressed	MILP + meta-heuristic	Higher execution time for large networks.
Gouareb *et al.* (2018)	Yes	Linear + delay-sensitive	Not addressed	MILP + K-shortest path and randomized algorithm	The VNFs are shared between the tenants but the chaining is not part of the optimization.
Alleg *et al.* (2017)	Yes	Linear + delay-sensitive	Not addressed	MILP	Works only for offline requests.
Fischer *et al.* (2019)	Yes	Linear + delay-sensitive	Not addressed	Meta-heuristic	Excessive execution time.
Ye *et al.* (2019)	No	Linear + delay-sensitive	Not addressed	Queueing model	Models only simple service function chains.
Ours	Yes	Arbitrary + delay-sensitive	VNFs are selected according to their KPIs	MILP + meta-heuristic	VNFs are not shared between tenants.

3.6 VNF Selection Problem

In this section, we start by listing the terms pertaining to this problem. Then, we present the problem definition before diving into the mathematical model. Table 3.2 summarizes the different terms and variables employed by the VNF selection model.

3.6.1 Problem definition

Given a virtual network service request r composed of a set of VNFs I_{r_k}, their desired KPIs, a set of vendors with their VNFs J_{v_l} and KPIs :

1) Select the minimum number of VNF vendors A_v;
2) Choose the best VNFs that match the desired KPIs of the VNFs I_{r_k} comprising a request r;
3) The selected VNFs J_{v_l} should be of the same type as the VNFs I_{r_k};
4) The selected VNFs J_{v_l} should not be shared among the set of virtual network services R for performance and security reasons.

We assume that the network/service provider has a myriad of VNFs from different vendors stored in its VNF repository to choose from by executing our proposed model and solutions, as depicted in figure 3.1. The KPIs could embody the desired resiliency level, throughput and response time, etc. . Finally, although the considered problem is off-line, we can easily use the model and the proposed solutions to solve the on-line problem by handling the requests R one at a time upon their arrival, for instance, following a FIFS (First-In First-Served) policy.

3.6.1.1 Decision Variables

Our decision variables are defined as follows : A_v equals 1 if a VNF vendor v is selected, and 0 otherwise. $Y_{j,i}^{v,r}$ equals 1 if VNF j from VNF vendor v is candidate for VNF i of request r, and 0 otherwise.

Table 3.2 Formulas notation

R	Set of virtual network service requests where, R= $\{r_1, r_2, ..., r_k\}$
V	Set of vendors, where V= $\{v_1, v_2, ..., v_l\}$
I_{r_k}	Set of VNFs of request r_k, where I_{r_k}= $\{vnf_1, vnf_2, ..., vnf_n\}$
E_{r_k}	Set of virtual links of a request r_k with complex policy and non linear topology
E_p	Set of physical links of the underlying network infrastructure
V_p	Set of physical nodes of the underlying network infrastructure
J_{v_l}	Set of VNFs, with different KPIs and configurations, available from vendor v_l where J_{v_l} = $\{\widehat{vnf_1}, \widehat{vnf_2}, ..., \widehat{vnf_m}\}$
D_{ij}	Weighted Euclidean distance between VNF i and VNF j
Θ_p, Θ_x	Represents the weight on server attribute p and the operational cost of server x, respectively
$Req_u^{\epsilon}, Rem_x^{\epsilon}$	Required/remaining amount of resources of VNF u/server x for resource type ϵ, which could be CPU, Memory and Storage capacity respectively
$C^{(x,y)}$, $\delta_{(x,y)}$	Represents the capacity and delay on link (x, y) respectively
T_i	Type of VNF i
$\theta^{(u,v),max}$, $Bw^{(u,v),r}$	Represents the maximum end-to-end delay and bandwidth requirements of virtual link (u, v) respectively
$KPI_r^{i,\alpha}, KPI_v^{j,\alpha}$, ω_α	Vector of attributes α desired by VNF i of request r and vector of attributes α offered by VNF j of vendor v respectively. The KPIs could be the desired resiliency level, throughput and response time, etc.. ω_α is a weight applied to each KPI attribute
$Y_{j,i}^{v,r} \in \{0, 1\}$	$Y_{j,i}^{v,r}$ = 1 if VNF j of VNF vendor v is candidate for VNF i of request r
$A_v \in \{0, 1\}$	A_v = 1 if vendor v is selected
$\alpha_{u,x}^r \in \{0, 1\}$	$\alpha_{u,x}^r$ = 1 if server x can host VNF u of virtual network service r, and 0 otherwise
$\beta_{(x,y)}^{(u,v),r} \in \{0, 1\}$	$\beta_{(x,y)}^{(u,v),r}$ = 1 if physical link (x, y) can carry the traffic of virtual link (u, v) belonging to virtual network service r, and 0 otherwise

3.6.1.2 Objective Functions

1) F_1 stands for selecting a small subset of VNF vendors. The aim is twofold : Use a maximum number of VNFs of the same VNF vendor and avoid having one VNF vendor per VNF for a

given request r. This could be for compatibility purposes or performance reasons;

$$\mathrm{F}_1 = \sum_{\forall v \in V} A_v \tag{3.1}$$

2) F_2 stands for finding the best matching between the VNFs of VNF vendors and VNFs of the whole request r. The aim is to choose, from the VNF vendors, the VNFs with KPIs that are close enough to the ones expected by the set of VNFs of the request r.

$$\mathrm{F}_2 = \sum_{\forall r \in R} \sum_{\forall v \in V} \sum_{\forall i \in I_r} \sum_{\forall j \in J_v} Y_{j,i}^{v,r} \times D_{ij} \tag{3.2}$$

D_{ij} is defined as the Weighted Euclidean distance between the KPI vector of VNF i and KPI vector of VNF j from VNF vendor v, where:

$$D_{ij} = \sqrt{\sum_{\forall \alpha \in [1..x]} \omega_\alpha \times (KPI_v^{j,\alpha} - KPI_r^{i,\alpha})^2} \tag{3.3}$$

ω_α is a weight applied to each KPI attribute.

3.6.1.3 Constraints

Constraints (3.4, 3.5, 3.6 and 3.7) ensure that exactly one VNF is selected amongst the entire pool of candidate VNFs from all VNF vendors. For a VNF i of a request r, a VNF i gets at most one VNF j from the pool of VNFs of VNF vendor v and a VNF j is assigned at most to one VNF i amongst all the requests R. These constraints, for the purpose of consistency, help the solver to not perform blindly, during the search for a solution, when selecting the VNFs and

assigning them to the requested VNFs with respect to their types and requirements.

$$\forall i \in I_r, \forall r \in R : \sum_{v \in V} \sum_{j \in J_v} Y_{j,i}^{v,r} = 1 \tag{3.4}$$

$$\forall v \in V, \forall j \in J_v, \forall r \in R : \sum_{i \in I_r} Y_{j,i}^{v,r} \leq 1 \tag{3.5}$$

$$\forall v \in V, \forall i \in I_r, \forall r \in R : \sum_{j \in J_v} Y_{j,i}^{v,r} \leq 1 \tag{3.6}$$

$$\forall i \in I_r, \forall v \in V : \sum_{r \in R} \sum_{j \in J_v} Y_{j,i}^{v,r} \leq 1 \tag{3.7}$$

Constraint 3.8 ensures that if a VNF j is candidate it means it comes from the selected VNF vendor v.

$$\forall i \in I_r, \forall v \in V, \forall r \in R, \forall j \in J_v : Y_{j,i}^{v,r} \leq A_v \tag{3.8}$$

Constraint 3.9 ensures that, for each request r, the type of VNF j offered by vendor v is like the type of VNF i.

$$\begin{aligned} &\forall i \in I_r, \forall v \in V, \forall r \in R, \forall j \in J_v : \\ &\qquad Y_{j,i}^{v,r} \times T(j) = Y_{j,i}^{v,r} \times T(i) \end{aligned} \tag{3.9}$$

Constraint 3.10 is for the positivity constraint on the decision variables.

$$\forall i \in I_r, \forall v \in V, \forall r \in R, \forall j \in J_v : A_v \in \{0,1\}, Y_{j,i}^{v,r} \in \{0,1\} \tag{3.10}$$

3.7 VNF Placement and Chaining Problem

In this section, we briefly describe the mathematical model we used for ARTIMIS. A complete description is available in Laaziz *et al.* (2019) for further details.

3.7.1 Objective Functions

Akin to Laaziz *et al.* (2019), we adopt a multi-objective approach and borrow two objective functions F_1 and F_3, as defined previously. We redefine them here as F_3 and F_4 respectively. Our focus in this paper is on minimizing operational costs and maximizing resource utilizations, which are of paramount importance for cloud and services providers since they have an impact on the OPEX (OPerational Expenditure). However, our approach is tailored for delay-sensitive virtual network services.

Nevertheless, other objective functions may be considered, requiring a slight adaptation to the mathematical model. Below are the studied objective functions:

$$\mathrm{F}_3 = Min \sum_{r} \sum_{u} \sum_{x} \alpha_{u,x}^{r} \times \Theta_x \tag{3.11}$$

$$F_4 = Min \left[\sum_{\boldsymbol{r}} \left[\sum_{u} \sum_{x} \alpha_{u,x}^{\boldsymbol{r}} \sum_{p \in P} \boldsymbol{\Theta_p} \left[1 - \frac{Req_u^p}{C_x^p} \right]^2 \right] \right] \tag{3.12}$$

3.7.2 Decision Variables

$\alpha_{u,x}^{r}$ realizes the placement of VNFs. It equals 1 if server x can host VNF u of virtual network service r, and 0 otherwise. $\beta_{(x,y)}^{(u,v),r}$ performs their chaining, and equals 1 if physical link (x, y) can carry the traffic of virtual link (u, v) belonging to virtual network service r, and 0 otherwise.

3.7.3 Constraints

$$\forall x \in V_p : \sum_r \sum_u \alpha_{u,x}^r \times Req_u^{CPU} \leq Rem_x^{CPU} \tag{3.13}$$

$$\forall x \in V_p : \sum_r \sum_u \alpha_{u,x}^r \times Req_u^{Memory} \leq Rem_x^{Memory} \tag{3.14}$$

$$\forall x \in V_p : \sum_r \sum_u \alpha_{u,x}^r \times Req_u^{Storage} \leq Rem_x^{Storage} \tag{3.15}$$

$$\forall (x,y) \in E_p : \sum_r \sum_{(u,v)} \beta_{(x,y)}^{(u,v),r} \times Bw^{(u,v),r} \leq C^{(x,y)} \tag{3.16}$$

$$\forall (u,v) \in E_{r_k}, \forall r \in R : \sum_{(x,y)} \beta_{(x,y)}^{(u,v),r} \times \delta_{(x,y)} \leq \theta^{(u,v),max} \tag{3.17}$$

$$\forall (u,v) \in E_{r_k}, \forall (x,y) \in E_p, \forall r \in R : \alpha_{u,x}^r + \alpha_{v,y}^r \leq \beta_{(x,y)}^{(u,v),r} + 1 \tag{3.18}$$

$$\forall (u,v) \in E_{r_k}, \forall (x,y) \in E_p, \forall r \in R : \alpha_{u,x}^r + \alpha_{v,y}^r + \beta_{(x,y)}^{(u,v),r} \leq 2 \tag{3.19}$$

$$\forall (u,v) \in E_{r_k}, \forall x \in V_p, \forall r \in R : \sum_y \beta_{(x,y)}^{(u,v),r} - \sum_y \beta_{(y,x)}^{(u,v),r} = \alpha_{u,x}^r - \alpha_{v,x}^r \tag{3.20}$$

$$\forall u \in I_{r_k}, \forall r \in R : \sum_x \alpha_{u,x}^r = 1 \tag{3.21}$$

$$\forall u \in I_{r_k}, \forall x \in V_p, \forall r \in R : \alpha_{u,x}^r \in \{0,1\}, \beta_{(x,y)}^{(u,v),r} \in \{0,1\} \tag{3.22}$$

Constraints 3.13, 3.14 and 3.15 refer to the limitation of physical resources of servers, requiring that they not be exceeded when deploying the VNFs. Constraint 3.16 states that the capacity of each physical link should not be exceeded by the traffic of all VNF pairs flowing on it. Constraint 3.17 enables to consider the end-to-end delay of complex virtual network topologies, i.e., not just linear topologies and small sizes with 3 to 4 VNFs. It checks whether or not each delay of a virtual link is met in order to satisfy the QoS requirement specified in the SLA. Constraints 3.18 and 3.20 enforce the joint placement and chaining of VNFs of each virtual network service.

Constraint 3.19 avoids allocating physical links for VNFs deployed on the same host for each virtual network service. Constraint 3.21 guarantees that a VNF instance is placed only on one server. Constraint 3.22 is for the positivity constraint on the decision variables.

3.8 ARTIMIS : Exhaustive Search based Algorithm

As the name implies, the exhaustive search-based algorithm examines all the VNFs within the VNF repository according to the requested VNFs' KPI and returns the VNFs that match the desired VNFs' KPI for the service function chain request. Algorithm 3.1 iterates over all the service function chain requests (line 4), and for each VNF (line 5), it selects the VNFs of the same type (lines 6 and 7). Then, it computes the weighted Euclidean distances (lines 8 and 9) given in equation (3.3) and sorts them using the Tim Sort algorithm in increasing order (line 11). In line 12, we assign the VNF with the minimal distance to the requested VNFs' KPI. Finally, we check if all the VNFs are handled (lines 14 to 16) and reject the request if there is no available VNF (line 17).

3.9 ARTIMIS : Non Dominated Sorting based Genetic Algorithm

Population Initialization. According to Gen & Runwei Cheng (1999), a population may be set up either deterministically or randomly. We randomly initialize the chromosomes by selecting the potential VNFs according to their types from the VNF repository. Algorithm 3.2 describes this random initialization. It iterates over all the genes, i.e., the VNFs of the request (line 2), and then it picks a VNF from the repository of the same type (lines 3 and 4) as the VNF of the request and assigns it to its corresponding gene (line 5).

Genotype Encoding. One of the main issues in designing a successful Genetic Algorithm is the genotype encoding step. The representation of a solution to the VNF selection problem is pictured in figure 3.3. We map a set of VNFs with their requested types into a set of VNFs with the same types provided either by the same or different vendors. The locus level displays the set of VNFs with the requested types while the allele level embodies the potential VNFs with respect to their VNF vendors who propose them as a candidate solution.

Algorithm 3.1 Exhaustive search-based algorithm

Input: The set of vendors with their VNFs, the SFC requests with their VNFs and the weights.
Output: The VNFs from the vendors that match the desired VNF's KPI of the SFC requests

```
Preprocessing phase :
Scale(VNF KPI vectors of the set of vendors)
Store the distances
distances={}
Store the assigned vnfs per vendor for the SFC requests
assignments={}
for request in SFC requests do
    for vnf in request do
        type = getTypeOfVnf(request, vnf)
        vnfsOfSameType=getVnfsOfSameType(setOfVendors, type)
        Get the Weighted Euclidean Distances
        for vnfType in vnfsOfSameType do
            distances[vnfType]=WED(vnfType, vnf, weights)
        end for
        sort in increasing order
        sortedDistances=SortInIncreasingOrder(distances)
        set the minimal distance
        assignments[request,vnf]=sortedDistances[0]
    end for
    Check if all the vnfs are handled and update the set of vendors
    if size(assignments[request]=size(request)) then
        accepted=accepted+1
    else
        rejected=rejected+1
    end if
end for
return assignments
```

NSGA-II generates a plurality of non-dominated solutions. We choose the non-dominated solution with the maximum number of satisfied VNFs since we could then have a plurality of non-dominated solutions.

Genetic operators. The following algorithms define the crossover and mutation operator procedures used by our genetic algorithm. For crossover, we apply the 1-cut point mechanism

Algorithm 3.2 Generation of chromosome

Input: The set of vendors with their VNFs, the SFC requests with their VNFs and the weights.
Output: A genotype
All the genes are set to be ∅.
The size of the Genotype equals the number of VNFs of a request.
1: Genotype=∅
2: **for** *gene* in Genotype **do**
3: vnfsOfSameType=**getVnfsOfSameType**(setOfVendors, gene.type)
Pick randomly a vendor and its VNF
4: vendorAndVnf=**random**(vnfsOfSameType)
5: **set**(gene, vendorAndVnf)
6: **end for**
7: **return** Genotype

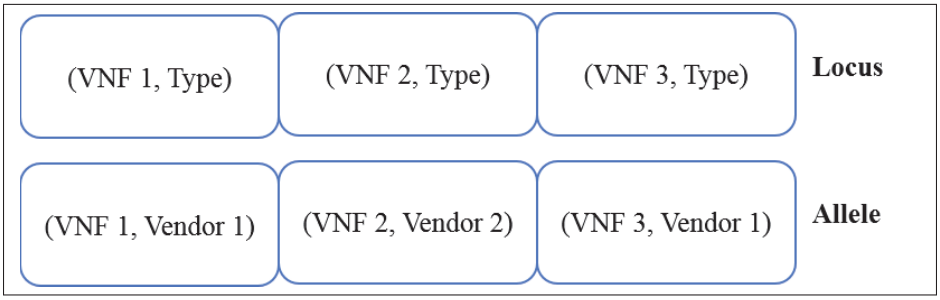


Figure 3.3 Chromosome encoding VNF selection problem

(Algorithm 3.3) and the mutation operator uses a custom-made mutation mechanism (Algorithm 3.4) that randomly identifies a locus and changes its allele. Basically, Algorithm 3.3 generates new chromosomes by reusing slices of the chromosomes already given as input, but first checks if they are valid (line 1). At the cut point (line 2), it takes the left part of the first chromosome and combines it with the right part of the second chromosome, thus generating the first child chromosome (line 3). Like this first child chromosome, the second child chromosome is generated using the same principle (line 4).

Moving to Algorithm 3.4, it applies a small perturbation on the chromosome by randomly taking a gene (line 1) and modifying its current VNF by replacing it with another one from the repository (lines 4 and 5).

Algorithm 3.3 1-cut point crossover operator procedure

Input: Two genotypes P1, P2
Output: Two genotypes C1, C2
If the two chromosomes are valid then proceed
if P1.valid=0 and P2.valid=0 **then**
  CutPoint=**random**(0, **size**(P1))
  Split the parent chromosomes at CutPoint gene position and generate two children
  C1=**LeftPart**(P1, CutPoint) ∪ **RightPart**(P2, CutPoint)
  C2=**LeftPart**(P2, CutPoint) ∪ **RightPart**(P1, CutPoint)
end if
return C1, C2

Algorithm 3.4 gene-based mutation operator procedure

Input: Genotype
Output: The mutated Genotype
position=**random**(0, **size**(Genotype))
Get the VNFs of same type
vnfsOfSameType=**getVnfsOfSameType**(setOfVendors, gene[position].type)
if vnfsOfSameType **then**
  vendorAndVnf=**random**(vnfsOfSameType)1
  set(gene[position], vendorAndVnf)
end if
return Genotype

3.10 ARTIMIS : A Chemical Reaction Optimization Meta-Heuristic

In this section, we describe our CRO-based approach that we designed for the placement and chaining of VNFs. It is worth mentioning that our proposed approach reuses same core of the mathematical model, presented in section 3.7, in terms of constraints and decision variables following the rules that drive this framework. We handle constraints by confining the search to the feasible solution space. In other words, we generate solutions without violating any constraint in both the initialization and iterations phases of the CRO. Should interested reader find more details about this optimization framework in Lam & Li (2010).

3.10.1 Solution Encoding

Our molecule encoding is presented in table 3.3. We used a grouping technique inspired by Falkenauer (1996) to present the placement of each VNF in the substrate network. For each server of the physical infrastructure, we specify the set of VNFs hosted on it by specifying their IDs. Each VNF in the input request is identified by a unique ID, as specified in 3.4, which presents the encoding of the input SFC request. As an initialization process, we use the First Fit (FF) algorithm to generate the molecules. The basic idea underlying this algorithm is that for each VNF, the heuristic attempts to host it on the first server still having enough resources in terms of its residual CPU cores. For each VNF pair in the input request, we compute the shortest path using Dijkstra's algorithm. If the placement guarantees the requested delay specified by the service function chain in the input, then we consider the solution valid and add it to the population, otherwise, the molecule is discarded, and we generate a new solution that respects both requirements in terms of resource availability and delay constraints. Once the initial phase is completed, it provides us with a set of feasible molecules that ensure high consolidation of physical resources as well as good delays over the links, thanks to Dijkstra's algorithm. The initial molecules undergo a series of operations or reactions that will help enhance the quality of the initial population and return sub-optimal solutions when the stopping criterion of the algorithm is met.

Table 3.3 SFC Input Requests

	SFC 1			**SFC 2**				**SFC 3**	
VNF Id	1	2	3	4	5	6	7	8	9
Requested CPU cores	2	6	5	4	7	2	1	3	2

Table 3.4 Molecule Encoding

	Server 1	**Server 2**	**Server 3**	**Sevrer 4**
Hosted VNFs Ids	1,2	3,4	5,6,7	8,9
Total Used CPU cores	8	9	10	5

3.10.2 CRO Elementary Operators

- **Decomposition:** This reaction starts by randomly selecting a single molecule P from the population, and results in two different molecules C1 and C2. The aim of this procedure is to keep the best quality part of the initial solution and then correct the remainder, i.e., we maintain the server with the maximum used capacity from P and we copy its content into two new solutions (Steps 6 and 9 in Algorithm 3.5). We place the remaining VNFs that are not hosted on this selected server using the first fit algorithm on both new solutions. To ensure that C_1 is different from C_2 we vary the order of VNFs for each of the solutions. The process of this reaction is described in Algorithm 3.5.

Algorithm 3.5 CRO Decomposition

Input: One Molecule P_1
Output: Two Molecules C_1 and C_2
Decomposition:
1: **For** each server in P
2: Calculate the sum of allocated CPU
3: Rank the servers in a descending order based on their used CPU cores
4: **end For**
5: Choose the server with maximum used CPU S_1
6: Initialize a new solution C_1
7: Copy the content of S_1 to the first server of C_1
8: Reallocate the remaining VNFs in the new solution C1 by using the First Fit algorithm (The order of the VNFs to be placed is random)
9: Initialize a new solution C_2
10: Copy the content of S_1 in the first server of C_2
11: Reallocate the remaining VNFs in the new solution C1 by using the first fit algorithm
12: **if** All the pairs in C_1 and C_2 satisfy the delay constraints
13: **if** Fitness(C_1)>Fitness(P) **Or** Fitness(C_2)>Fitness(P)
14: Remove P from population
15: Add C_1 and C_2 to the population
16: **else**
17: Discard C_1 and C_2

- **On-wall ineffective collision:** This procedure also starts with a single input solution, P, but unlike the decomposition, it results in only one output solution. This reaction represents an improving step where we choose the least used server in the initial solution, P, and redistribute

its VNFs on the currently used servers if they are still capable of hosting other VNFs, and still have available resources. This reaction is described in detail in Algorithm 3.6.

Algorithm 3.6 CRO Placement -On-Wall Ineffective Collision

Input: One Molecule P
Output: One molecule C
On-wall Collision
Copy the solution P into C
For each server in C
  Calculate the sum of allocated CPU
  Rank the servers in a descending order based on their used CPU
end For
  Choose the server with minimum used CPU S_1
  Empty the selected server S_1
  Redistribute the previously hosted VNFs on S_1 using first fit algorithm
if All the pairs in C satisfy the delay constraints
if Fitness(C) > Fitness(P)
  Remove P from population
  Add C to the population
else
  Discard C

- **Synthesis:** In this reaction, we randomly select two distinct molecules P_1 and P_2 from the population and obtain a single molecule C. The goal of this reaction is to keep the best features of both initial solutions P_1 and P_2 in the resulting molecule C. We start by selecting the fullest servers on both P_1 and P_2 and and then copy their configurations into the new solution C. Before proceeding with the remaining VNFs, we must make sure, that there are no duplicate VNFs in these two servers, and then using the first fit algorithm, we store the rest of the VNF requests. The details of this reaction are described in Algorithm 3.7.
- **Inter-molecular ineffective collision:** In this procedure, we also select two random molecules P_1 and P_2 from the population and we get two output solutions C_1 and C_2. The CRO framework states that each new solution is constructed independently from the other one, and so we decided to use the on-wall ineffective operator twice to generate two new solutions: C_1=ON-WALL COLLISION(P_1) and C_2=ON-WALL COLLISION(P_2).

Algorithm 3.7 CRO Placement - Synthesis

Input: Two Molecules P_1 and P_2
Output: One molecule C
On-wall Collision
For each server in P_1
  Calculate the sum of allocated CPU
  Rank the servers in a descending order based on their used CPU
  Choose the server with maximum used CPU S_1
end For
For each server in P_2
  Calculate the sum of allocated CPU
  Rank the servers in a descending order based on their used CPU
  Choose the server with maximum used CPU S_2
end For
  Initialize a new empty solution C
  Copy the placed VNFs in S_1 and S_2 in two new servers of C
  Eliminate VNF duplicates between S1 and S2
  Look for the remaining unplaced VNFs in the SFC request
  Place those VNFs using First Fit algorithm in the solution C
if All the pairs in C satisfy the delay constraints
if Fitness(C) > Fitness(P_1) **Or** Fitness(C) > Fitness(P_2))
  Remove P_1 and P_2 from population
  Add C to the population
else
  Discard C

3.11 Asymptotic Analysis

Now, we describe the algorithmic complexity of our proposed approaches used by ARTIMIS. It is worth mentioning that this complexity analysis is performed in its worst-case scenario. We start with the exhaustive search based algorithm. Its complexity is $O(|R| \times |I_{r_k}| \times |J_{v_l}| \log |J_{v_l}|)$. Here, $|R|$ is the number of requests and $|I_{r_k}|$ is the number of VNFs that compose a request $r_k \in R$. $|J_{v_l}|$ is the number of VNFs within the set of VNFs. Since we use Tim Sort algorithm to sort the VNFs, its complexity is $O(|J_{v_l}| \log |J_{v_l}|)$. The complexity of NSGA-II is known to be $O(M \times N^2)$. Here, N is the population size and M is the number of cost functions. Approximately, the global complexity of this approach is $O(N^2)$. Similarity, the Pareto-based Dominance is

$O(N^2)$. Moving to the chemical reaction optimization operators, the solution encoding executes in $O(|(I_{r_k})|^2) + (|V_p| + |E_p| \log |V_p|)$ since we make use of First Fit and Dijkstra's algorithms. Algorithm 3.5 executes in $O(|(I_{r_k})|^2) + (|V_p|^2 \times \log |V_p|)$ as we use Tim Sort algorithm to sort the servers. Similarly, Algorithm 3.6 runs in $O(|(I_{r_k})|^2) + (|V_p|^2 \times \log |V_p|)$. Algorithm 3.7 runs in $O(|(I_{r_k})|^2) + (|V_p|^2 \times \log |V_p|) + (|V_p| + |E_p| \log |V_p|)$. Finally, the complexity of our chemical reaction optimization procedure for the joint placement and chaining of VNFs is $O(MaxIter \times |(I_{r_k})|^2) + (|V_p|^2 \times \log |V_p|) + (|V_p| + |E_p| \log |V_p|)$ where $MaxIter$ is the maximum number of iterations.

3.12 Evaluation

In this section, we evaluate the core of ARTIMIS, i.e., the VNF selection with the joint placement and chaining of VNFs procedures. We assess our chemical reaction optimization algorithm both on a small- and large-scale networks. We evaluate the quality and effectiveness of the solution with respect to the optimal solution and against a novel state-of-the-art cultural genetic algorithm we proposed in Laaziz *et al.* (2019) which runs on top of a non dominated sorting based genetic algorithm.

3.12.1 Setup

We used Gurobi 7.5.1 to get the optimal solution. The VNF selection procedure was implemented using Python 2.7, and the deployment procedure was implemented using Java 8.0. The experiments and implementations were carried out on a physical machine composed of 8 CPU cores. It is worth mentioning that the Pareto-Dominance and NSGA-II based approaches, for the selection stage, were implemented and adapted to our problem following the guidelines provided in Platypus David Hadka (2015). We use the same settings as in Laaziz *et al.* (2019), for the deployment stage, and the parameters of the CRO-based approach were set on a trial and error process.

We conduct our experiments on different types of topologies, i.e., small-scale, medium-scale and large-scale substrate networks composed of 20, 40, 60, 80 and 120 physical nodes, respectively. These substrate networks were generated using the NetworkX library, following the same sight line of Laaziz *et al.* (2019), Pham *et al.* (2020), Mechtri *et al.* (2016), Khebbache *et al.* (2017), Khebbache *et al.* (2018), Fischer *et al.* (2019), to study the performance of CRO, regarding the deployment of virtual network services. We report the average values from the experiments which were repeated 10 times. Finally, tables 3.5 and 3.6 summarize the characteristics of the available VNFs for the selection procedure, and the different characteristics of the virtual network services, in addition to the KPIs of VNFs prior to executing the CRO-based approach, respectively.

Table 3.5 Characteristics of the available VNFs

Resiliency level	[99.0, 99.9, 99.99, 99.999, 99.9999, 99.99999, 99.999999]
Throughput (Mbps)	Between 1000 and 5000
Response time (ms)	Between 10 and 50
Number of VNFs	1000 different types (e.g., firewall, IDS, NAT, IPS) and configurations provided by 2 vendors

Table 3.6 Characteristics of the virtual network service

Number of requests	varied from 5 to 45
Number of VNFs per request	10
Desired resiliency level	99.99
Desired throughput (Mbps)	1500
Desired response time (ms)	35
KPI weights (response time, throughput, resiliency level)	[1,1,5]

3.12.2 Performance Metrics

We evaluate ARTIMIS, i.e., the VNF selection and the joint placement and chaining of VNF mechanisms, according to the following metrics to assess their performances. First, we present the performance metrics related to the VNF selection procedure. Second, we present the performance metrics used to compare the proposed CRO with CGA Laaziz *et al.* (2019) and ILP.

3.12.2.1 VNF Selection Procedure

1) ***Matching level*** : We calculate the ratio of satisfied VNFs in terms of KPI. Basically, it estimates the number of VNFs returned by the selection procedure that match the requested VNFs with respect to the KPIs;
2) ***Runtime*** : We compute the time taken by the model/meta-heuristic to converge to a solution (optimal/close to optimal);
3) ***Rejection rate*** : We quantify the ratio of rejected service requests due to non-identical VNFs offered by vendors.

3.12.2.2 VNFs Placement and Chaining Procedure

1) ***Resource utilization*** : We evaluate the resource utilization on the servers in terms of CPU cores. It is worth mentioning that different dimensions may be considered as well, such as Memory and Storage capacity;
2) ***Operational cost*** : We quantify the average operational cost in terms of number of servers used to provision the virtual network services;
3) ***Runtime*** : We compute the time taken by the ILP model/meta-heuristic to converge to a solution (optimal/close to optimal);
4) ***Accepted requests*** : We estimate the total number of requests that are accepted;
5) ***End-to-end delay*** : We determine the maximum end-to- end delay per pair of VNFs;
6) ***SLO violation*** : We measure the ratio of requests that experienced SLO violations.

3.12.3 Experiments

3.12.3.1 VNF Selection Procedure

In the following experiments, we evaluate the proposed approaches for the VNF selection procedure. Here, we consider a virtual network service composed of 10, 15 and 20 VNFs, respectively while varying the number of requests from 5 to 45.

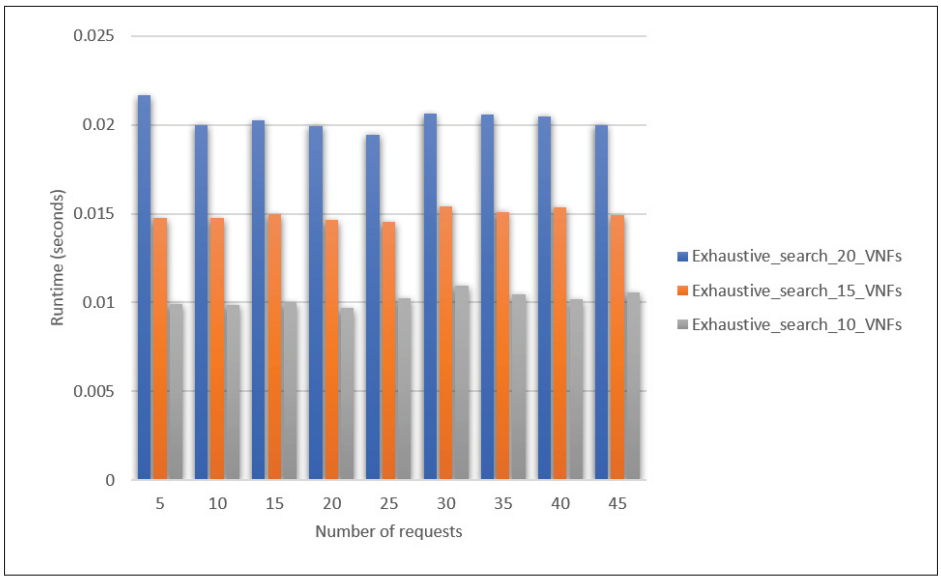


Figure 3.4 Average runtime per accepted request as a function of number of VNFs while varying the number of requests

The results of the average runtime per accepted request for the exhaustive search algorithm, under different virtual network request sizes, are shown in figure 3.4. We can clearly see that increasing the number of VNFs from 10 to 15 and from 15 to 20 increases the runtime by ≈ 5 milliseconds. This result is due to the logarithmic complexity of the exhaustive search selection procedure, as previously stated in the asymptotic analysis section. Consequently, the algorithm exhibits a relatively short runtime regardless of the request size in terms of VNFs.

Figure 3.5 describes the matching level achieved by the exhaustive search algorithm, under different virtual network request sizes. In this experiment, the number of vendors is 1, for the

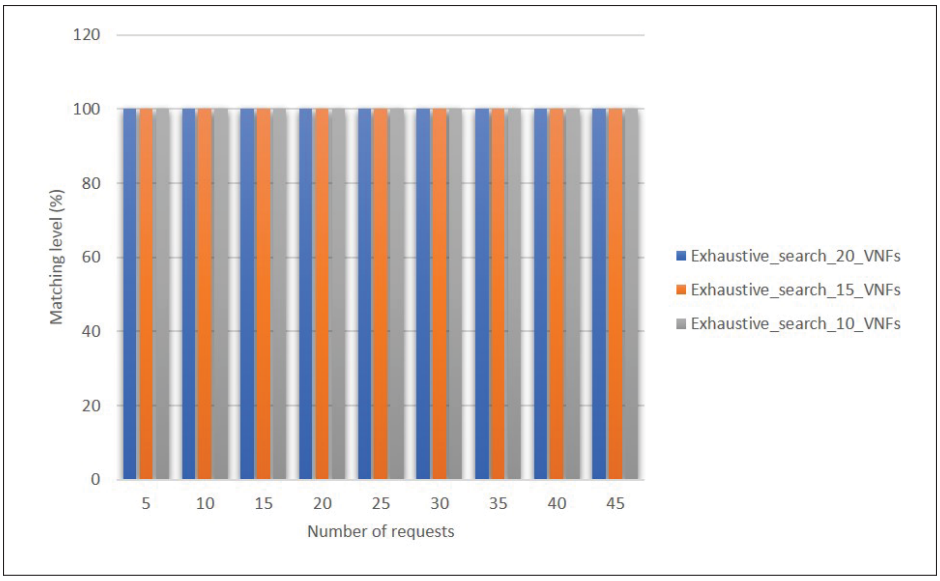


Figure 3.5 Matching level as a function of number of VNFs while varying the number of requests

sake of simplicity and following the network scenario defined in section 5.2, but the model is defined to account for different vendors, as described in equation 1. From figure 3.5, we can clearly see that regardless of the size of the request, our algorithm is able to achieve a perfect matching level ≡ 100 % since it performs an exhaustive search among all the available VNFs' KPIs. Therefore, our algorithm selects exactly the VNFs, with the desired KPIs, from the vendor, that match the VNFs' KPIs that were stated within the requests prior to their deployment.

Figure 3.6 compares the matching level of the exhaustive search algorithm, the ILP and the Pareto Dominance-based Genetic Algorithm with 10 VNFs under different virtual network request sizes. The results show that the ILP solution failed to achieve a 100 % score for more than 10 requests. As the number of requests exceeds 10, the matching level for the ILP approach increases as we increase the number of requests, and it varies between ≈ 90% and ≈ 98%. This result is attributable to the fact that compared to the exhaustive search algorithm and Pareto Dominance-based Genetic Algorithm, the ILP tries to achieve the minimum Weighted Euclidean distance over all the requests. Therefore, a perfect matching is not achieved at 100 % for some VNFs. Regarding the Pareto Dominance-based approach, it failed shortly to score a 100%

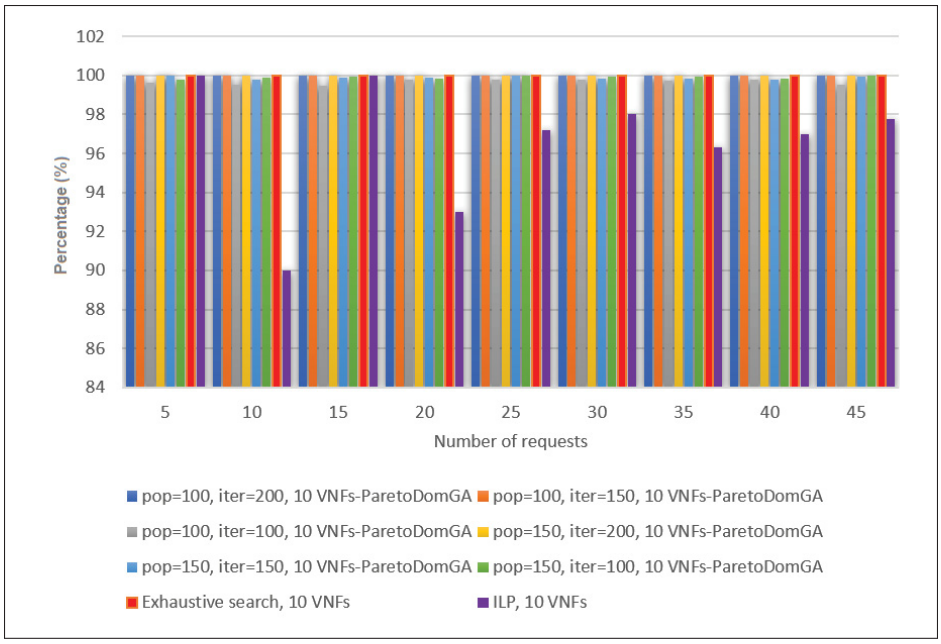


Figure 3.6 Matching level with 10 VNFs while varying the number of requests

(between 99.5% and 99.8%) like the exhaustive search-based approach with 150 individuals (both with 100 and 150 iterations). This suggests that, when tuned with 150 individuals, it performs well and the fitness of the chromosomes does not deteriorate over the iterations. However, with 100 individuals, the Pareto Dominance-based Genetic Algorithm was able to achieve the same matching level as the exhaustive search-based algorithm, both with 100 and 150 iterations. Thus, the exhaustive search-based algorithm and the Pareto Dominance-based Genetic Algorithm, with 150 individuals (both with 100 and 150 iterations) achieves a 100 % matching level.

Figure 3.7 illustrates the average runtime per accepted request resulting from the execution of the exhaustive search-based algorithm and the Pareto Dominance-based approach. We do not report the results of the ILP in terms of runtime as its computational complexity is known to be NP-Hard, which increases exponentially as we increase the number of virtual network requests. As can be seen from figure 3.7, the exhaustive search based algorithm exhibits smaller runtime 10 ms as compared to the Pareto Dominance Genetic Algorithm, regardless of the population size (100 and 150) and iterations (100, 150 and 200), which is between $\approx$ 0.4 s and $\approx$ 0.83 s

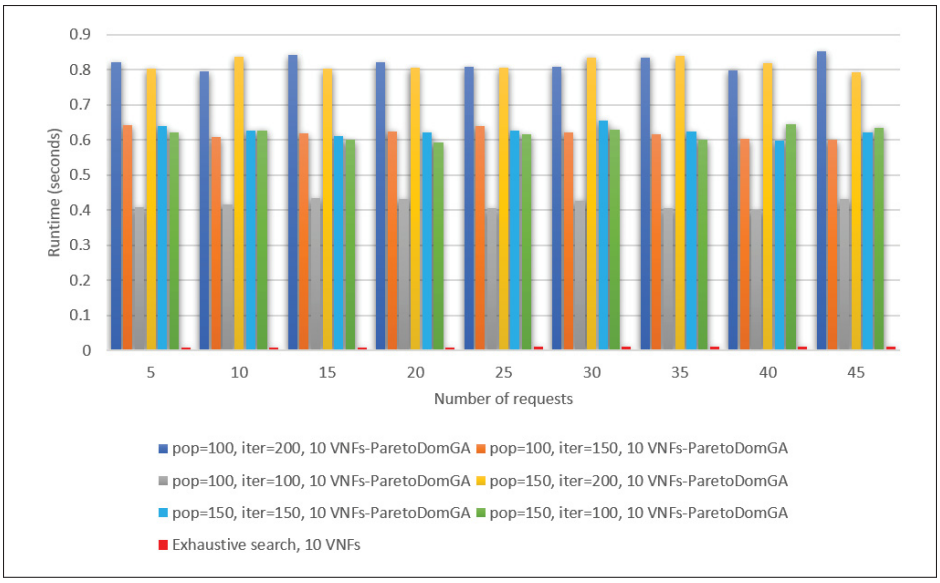


Figure 3.7 Average runtime as a function of number of VNFs while varying the number of requests

respectively. This result is since as we increase the number of iterations and population size, the runtime increases as well, since the algorithmic complexity of the Pareto Dominance Genetic Algorithm is mainly affected by these parameters.

Figure 3.8 reports the rejection rate obtained by the exhaustive search-based algorithm, the Pareto Dominance-based Genetic Algorithm and the ILP, with 10 VNFs under different virtual network request sizes. From the figure, we can clearly see that the approaches are able to accept all the requests since they select the corresponding VNFs with the desired KPI from the set of available VNFs. Mention should be of the data points that are overlapping with each other as the rejection rates are all zeros. Indeed, the rejection rate will depend on the VNFs offered by the vendor (the set J_{v_l}). In this test experiment, we consider that an identical VNF requested by r is offered by vendors to assess the capacity of the approaches to find the requested VNFs. Nonetheless, these approaches may be adapted to take into consideration non-identical VNF scenarios, i.e., when there is no perfect match.

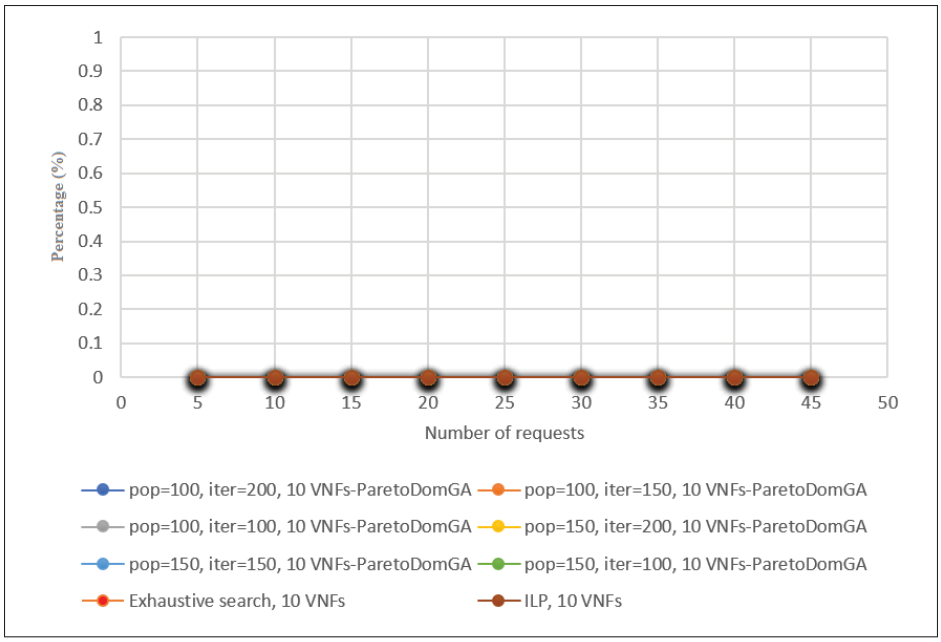


Figure 3.8 Rejection rate as a function of the number of VNFs while varying the number of requests

3.12.3.2 VNF Placement and Chaining

In the following experiments, we compare our CRO-based approach against state-of-the-art approaches Laaziz *et al.* (2019). That is, we benchmark it with ILP and CGA.

Figure 3.9 illustrates the percentage of the remaining CPU cores for all the accepted requests in each network graph obtained following the execution of the different approaches. We can see that CGA and CRO behave in a similar fashion for the different network infrastructures. This observation suggests that the approaches can efficiently use the available resources, as they are very close to to the ILP even though they do not have the same number of accepted requests, as can be seen in figure 3.11.

Figure 3.10 shows the total runtime for all the accepted requests in each network graph resulting from the different approaches. It can clearly be seen that the total runtime for the ILP solution increases exponentially as compared to CRO and CGA, respectively, with respect to the size of the network infrastructure. For example, for 120 servers, ILP takes $\approx$ 26 minutes to deploy all the

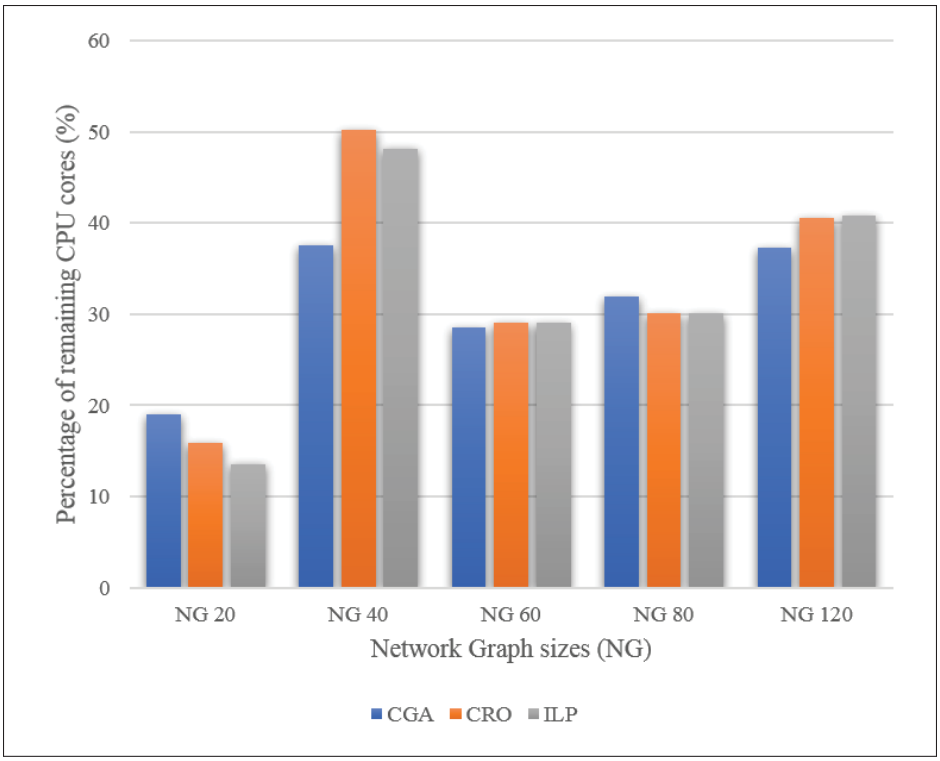


Figure 3.9 Average remaining CPU cores as a function of NG sizes

requests, while CRO and CGA need ≈ 13 seconds and ≈ 8 seconds, respectively. CRO exhibits the smallest total runtime for small-scale networks, whilst CGA is more suitable for medium-and large-scale networks (80 and 120 servers). It is obvious from the complexity analysis section that the number of VNFs, the size of the substrate network, and the parameters of CGA and CRO have a direct impact on their runtime. From this evaluation, we can state that the decision maker has the choice of selecting the appropriate approach to fit its need, whether over a small- or medium to large-scale network infrastructures, while being satisfied with the quality of the solutions obtained.

Figure 3.11 illustrates the total number of accepted requests in each network graph resulting from the different approaches. It can clearly be seen that the ILP solution accepts more requests as compared to CRO and CGA, respectively. This is because ILP does exploit the whole search space, i.e., it examines all the available resources efficiently as it never deploys VNFs on other

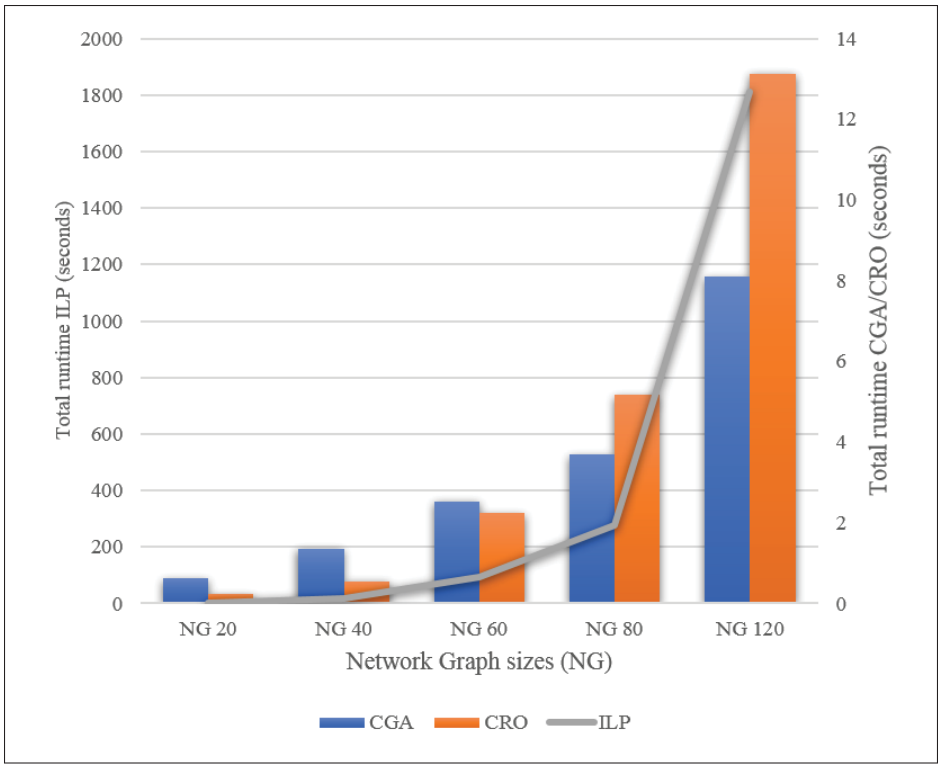


Figure 3.10 Total runtime as a function of NG sizes

servers unless a server is fully used. Overall, the gap, between CRO and CGA is very small ($\approx$ 1 to 5 requests), as compared to ILP ($\approx$ 10 requests), which suggests that they can effectively be used and integrated into NFV orchestration systems to help decision makers efficiently exploit their infrastructures and effectively deploy virtual network services in a reasonable amount of time.

Figure 3.12 shows the total number of servers used in each network graph experienced by the different approaches, namely, the ILP, CRO and CGA respectively. CRO is very competitive regardless of the substrate network size as compared to CGA and exhibits a similar behavior with respect to ILP. As an example, CRO enables the use of $\approx$ 10 servers less than CGA with a substrate network composed of 120 nodes. This is because CRO, for this metric, can properly enforce the objective function F3 and F4. However, for all the approaches considered, increasing the number of virtual network requests has a direct effect on any increase in the number of servers used, as more servers are needed to accommodate them.

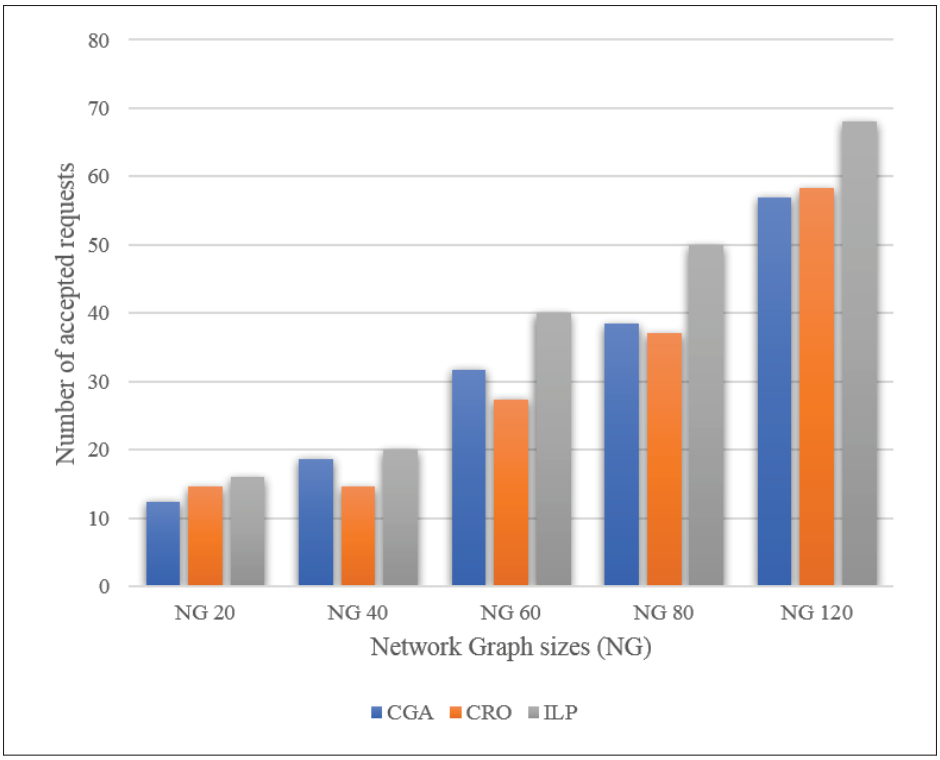


Figure 3.11 Average number of accepted requests as a function of NG sizes

Figure 3.13 illustrates the maximum delay per pair of VNFs in each network graph obtained by the different approaches. We can see that all the approaches, i.e., CRO, CGA and ILP, are able to deploy the virtual network requests while guaranteeing fulfilling the desired end-to-end delay. This is because the end-to-end delay is considered as a constraint rather than a cost function to be minimized, which allows the decision maker to fulfill the expressed end-to-end delay that must be met for all requests. It can be clearly seen be that this end-to-end delay is within ≈ 80 ms, and all the approaches perform well and very close to the ILP.

Figure 3.14 shows the ratio of requests with SLO violations in each network graph experienced by the different approaches for all the deployed requests. It can clearly be seen that the solutions provided by the different approaches, namely, CRO, CGA and ILP, satisfy the requirements of the accepted requests in terms of end-to-end delay, since this delay is considered as a constraint rather than a cost function to be minimized. Moreover, the two cost functions we defined ensure

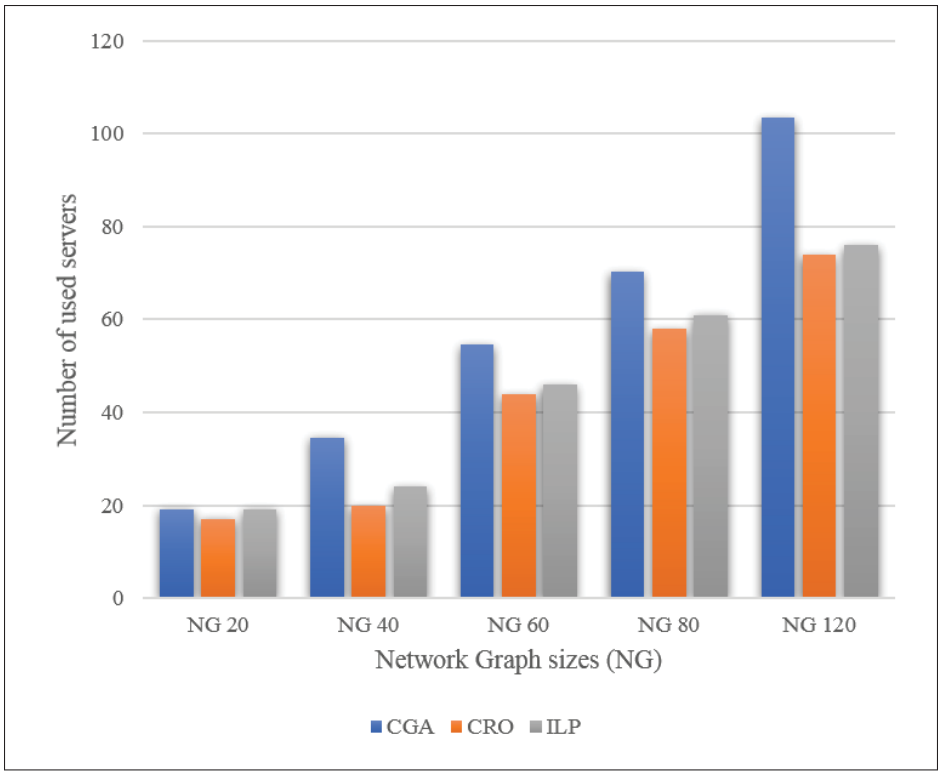


Figure 3.12 Average number of used servers as a function of NG sizes

that the VNFs are being deployed within the vicinity of each other, guaranteeing the end-to-end delay and at the same time fully exploiting the server's available resources. Thus, our approaches enforce the SLO.

3.12.4 Summary and Analysis

We now provide, from a bird's-eye view, a synthesis of how our approaches performed in terms of the VNFs selection procedure prior to their placement and chaining.

1) **VNF selection procedure :**

- The exhaustive search algorithm, in terms of convergence time and matching level, outperforms the other approaches, i.e., ILP and the Pareto Dominance-based Genetic Algorithm, regardless of the number of VNFs per virtual network service. In fact,

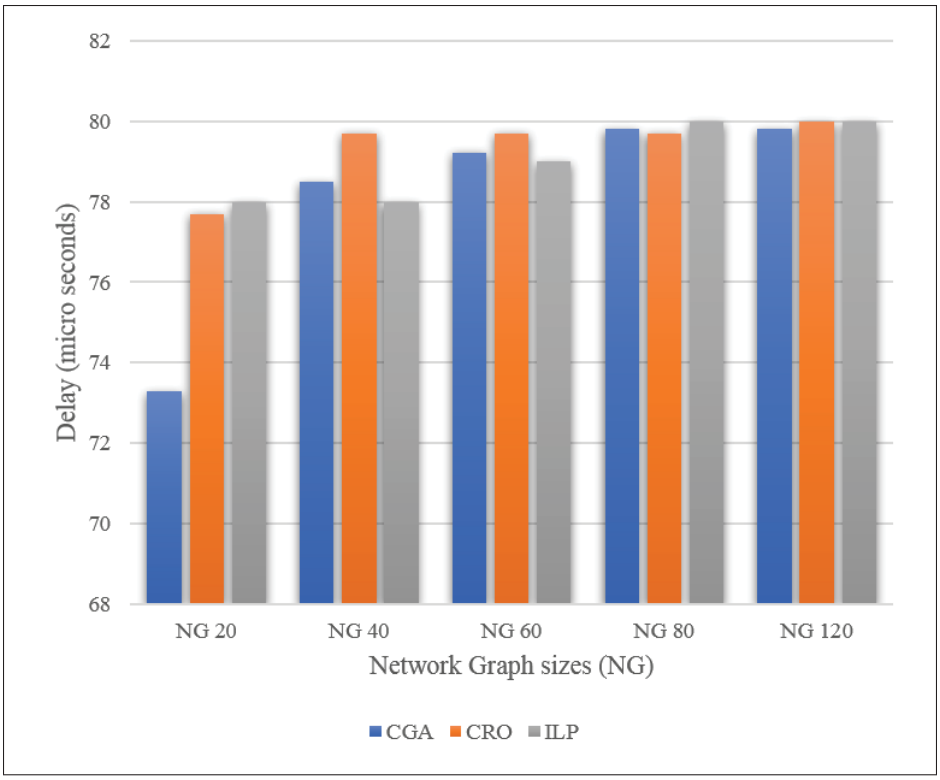


Figure 3.13 Average maximum delay as a function of NG sizes

for 20 VNFs, the exhaustive search achieves perfect matching in less than ≈ 1.2 ms, as compared to the Pareto Dominance-based Genetic Algorithm, which takes ≈ 0.6 seconds.

- The proposed approaches, namely, the exhaustive search algorithm, ILP and Pareto Dominance-based Genetic Algorithm, help minimize the rejection rate.
- These approaches are scalable and able to handle different types of VNFs, different numbers of VNFs associated to vendors, and the size of the virtual network services in terms of VNFs.

2) **VNF placement and chaining procedure :**

- As compared to CGA and ILP, CRO allows reducing the number of nodes used by ≈ 35% and ≈ 5 % for medium and large network instances, respectively, but comes with the extra cost of fewer accepted requests

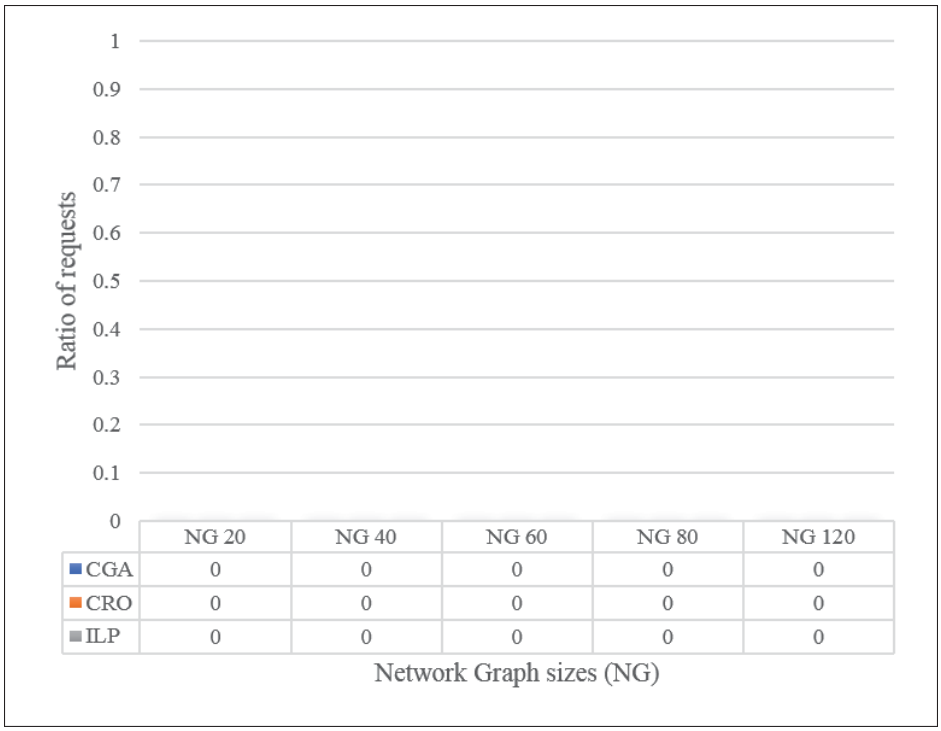


Figure 3.14 Average of SLO violation requests as a function of NG sizes

- CRO, ILP and CGA enable the enforcement of SLO for delay-sensitive virtual network services, regardless of the number of VNFs. Indeed, CRO keeps the delay below the maximum tolerated end-to-end value by enforcing Constraint 3.17.
- CRO is very competitive as compared to CGA and ILP in terms of quality of solutions.
- CRO is as effective as CGA and ILP in terms of average remaining CPU cores, which suggest that the approach exploits the search space efficiently by enforcing objective function F_4 regardless of the network infrastructure size.
- It is obvious, from the experiments, that CRO can deploy virtual network services quickly when the network infrastructure is small (e.g., networks composed of 20, 40 and 60 servers). As the size of the network grows (e.g., networks composed of 80 and 120 servers), CRO needs extra seconds, due to its algorithmic complexity but is better than CGA and as good as ILP in terms of used nodes.

3.13 Conclusion and Future Work

In this paper, we have presented ARTIMIS for the selection of VNFs and the dynamic deployment of delay-sensitive virtual network services. We first defined the system architecture, the system model whose main components consists of a VNF selection mechanism and a Chemical Reaction Optimization-based meta-heuristic procedure. The problems were formulated using an Integer Linear Program to get the optimal solution for small-scale sizes while meta-heuristics and an exhaustive search algorithm were proposed to deal with the scalability issues facing Integer Linear Programs. For the selection procedure, we proposed a set of technical solutions, namely, an exhaustive search-based algorithm and an evolutionary algorithm-based approach with the help of NSGA-II. For the placement and chaining procedure, a novel Chemical Reaction Optimization meta-heuristic approach was proposed and adapted to handle delay-sensitive virtual network services. Experiments with real-world network infrastructures show that our proposed approaches help achieve fast decision-making thanks to their low time complexities. In addition, the obtained solutions are of good quality as shown by the benchmark. In fact, CRO helps minimize the operational cost and achieve better resource utilization compared to CGA. However, CRO enables enforcing the SLO.

As future work, we will investigate the adaptation of the proposed model for non-perfect matching of VNFs proposed by vendors. In addition, we will consider the KPIs defined in the selection procedure prior to the placement and chaining of the VNFs (e.g., server availability). Furtheremore, we will investigate some pruning techniques to improve the computational complexity of CRO for large-scale infrastructures. Finally, adapting CRO by integrating the concept of Pareto Principle as in Bechikh, Chaabani & Said (2015) to have a fair comparison with Laaziz *et al.* (2019) from a multi-objective perspective.

Acknowledgments

This work has been supported by Ericsson Canada and the Natural Sciences and Engineering Research Council of Canada (NSERC).

CHAPTER 4

VALKYRIE: A SUITE OF TOPOLOGY-AWARE CLUSTERING APPROACHES FOR CLOUD-BASED VIRTUAL NETWORK SERVICES

Imane El Mensoum[1] , Laaziz Lahlou[1] , Fawaz A. Khasawneh[1] , Nadjia Kara[1] , Claes Edstrom[2]

[1] Département de Génie Mécanique, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3

[2] Ericsson, Canada,
8275 Rte Transcanadienne, Saint-Laurent, QC H4S 0B6

Paper submitted to IEEE Transactions on Services Computing on August 2020

4.1 Contributions

This is a joint research work in collaboration with Imane El Mansoum and Fawaz A. Khasawneh, a graduated master student and a postdoctoral research fellow. Imane developed the CRO-based clustering approach and Fawaz the Game Theory-based clustering method, which are described in sections 4.7 and 4.8, respectively.

4.2 Abstract

Complex networks are effective tools for modeling, studying and analyzing complex interactions between objects. In computer networks, these tools play an important role in understanding applications, end-users and interactions between compute nodes and their behaviors. Nowadays, computer networks are undergoing significant expansion due to the proliferation of network devices and compute nodes. One of the main challenges in computer networks lies in categorizing these compute nodes into clusters of connected compute nodes within these large-scale structures sharing similar features (e.g., CPU, memory, disk, etc.). In this paper, we propose a set of novel, dynamic and proactive topology-aware clustering approaches, namely, a MILP, a chemical reaction optimization and a game theory approaches, to form clusters based on the compute nodes' features and their topological structures. Our solutions are tailored to meet the requirements

of the fields of NFV and Cloud-based Networks. In this regard, the solutions aim to help decision makers facing issues related to scalability and computational complexities of their mechanisms to deploy their cloud-based services effectively. Experimental results demonstrate the effectiveness of the proposed approaches and their suitability, given their polynomial time complexities, which make them easy to integrate into cloud providers' orchestration systems compared to K-Means and DBSCAN.

Keywords : Network function virtualization, attributed network infrastructures, clustering, multi-objective optimization, topological structure, attribute similarity.

4.3 Introduction

Complex networks Bothorel, Cruz, Magnani & Micenková (2015) are formal and effective tools for modeling, studying and analyzing complex interactions between objects with non-trivial features in different domains. Examples are computer communication networks, brain networks and social networks. In computer networks, these tools play an important role in understanding applications, end-users and interactions between compute nodes and their behaviors. These tools essentially make use of the graph theory framework, where nodes represent objects and edges represent the interaction between nodes. In the context of computer networks, nodes represent commodity servers, compute nodes, network devices (legacy/virtualized), whereas edges embody their relationships, which can be diverse (e.g., dependencies, bandwidth capacity or latency) with respect to the context in which they are defined.

Clustering is a useful and important unsupervised learning technique widely used in the literature Baroni, Conte, Patrignani & Ruggieri (2017) Zhou, Cheng & Yu (2009) Cheng, Zhou & Yu (2011) Clauset, Newman & Moore (2004). It aims at grouping similar objects into one cluster while keeping dissimilar objects in separate clusters. Clustering has broad applications, including fraud detection and analysis of financial, time series, spatial and astronomical data, etc. Clustering of attributed graphs Zhou *et al.* (2009) represents an interesting challenge, which has

recently started a lot of attention. Graph clustering applications include areas such as community detection in social networks, etc.

Several approaches have been proposed to cluster attributed graphs. These approaches can be classified into two categories: parameter-free and parameter-dependent approaches. In a parameter-dependent approach, the number of clusters to be formed is given by the user as input for the clustering algorithm, in contrast to the parameter-free approach, where no such input is required.

In addition to the aforementioned classification, many existing clustering methods either perform clustering only considering the nodes' properties and/or topological structure. The choice of an approach depends mainly on the nature of the problem at hand and the desired goal. This choice is dictated by the nodes and/or by the links between them (i.e., focused on the structural part).

Generally, the clusters are formed by computing a similarity function considering either the node attributes and/or structural attributes. This similarity function is the key to building clusters since cluster members are grouped together only when being similar. In our clustering formulation and modelling, we will consider the quality of the clustering during the formation process, in addition to other cost functions to evaluate. Most existing approaches Bothorel *et al.* (2015), Zhou *et al.* (2009), Baroni *et al.* (2017) and Cheng *et al.* (2011) for the clustering of attributed graphs evaluate the quality of clustering once clusters are built. This is indeed how things are done by most traditional clustering approaches as well i.e. the quality of the clustering is evaluated at the end of the process, instead of quality being considered during the actual clustering. However, in our approaches, our aim is to attempt to form clusters by assessing or continuously improving them, considering the node properties and, given the heterogeneity and complexity of infrastructures, its structural aspects, hence being topology-aware. Indeed, these aspects should not be overlooked since the clustering logic is impacted and the resulting clusters could potentially diminish the performances of the applications and services when deployed and provisioned by the network or service provider.

In the present paper, we fill existing gaps in the research literature with the following main contributions:

1) We provide the first formulation for the joint server and network attributes for the dynamic and proactive clustering problem tailored for service function chains, and broadly, for virtual network services, in the context of NFV;
2) The problem is formulated as a Quadratic Constrained Integer Linear Program, implemented and solved in line with Gurobi, to find optimal solutions for small-scale networks;
3) We design a fast and scalable Chemical Reaction Optimization approach to handle medium and large-scale instances of the problem, leveraging the same ILP structure (cost functions and constraints) for reliable bench-marking;
4) Similar to item 1, the clustering problem is formulated as a matching game and solved using an adapted version of Irivng's algorithm Irving (1985);
5) Moreover, our proposed solutions easily integrate into orchestration systems following NFV MANO thanks to their low computational complexities;
6) VALKYRIE's performance in terms of solution quality and scalability is assessed using real-world topologies: small, medium and large scale enterprise networks;
7) Experimental results reveal that our proposed approaches exhibit better performance compared to traditional and well known clustering approaches K-means Bishop (2006) and DBSCAN Ester, Kriegel, Sander & Xu.

The rest of this paper is organized as follows. Section 4.4 presents the context, motivation and system architecture. The related works is discussed in Section 4.5. Section 4.6 presents the system model and state the clustering problem. Our clustering techniques based on Chemical Reaction Optimization and Game Theory are presented in section 4.7 and 4.8 respectively. Section 4.9 discusses their asymptotic analysis. The experimental evaluation is presented in Section 4.10, followed by the discussion of the results in Section 4.11. Finally, Section 4.12 concludes this paper.

4.4 Context, Motivation and System Architecture

Attributed graphs model real networks by augmenting their nodes with a set of attributes. In the field of networking, these attributes pertain to CPU, Memory, Storage capacity, Energy level, etc. . Thus, our clustering approach of an attributed graph is devoted to the service function chaining problem and broadly to cloud-based virtual network services where the goals are :

1) Reducing the search space from an algorithmic perspective to help find faster solutions for the algorithms we have developed to deploy them;
2) Build on-demand clusters such as CPU intensive clusters, bandwidth efficient clusters, etc. with respect to their requirements.

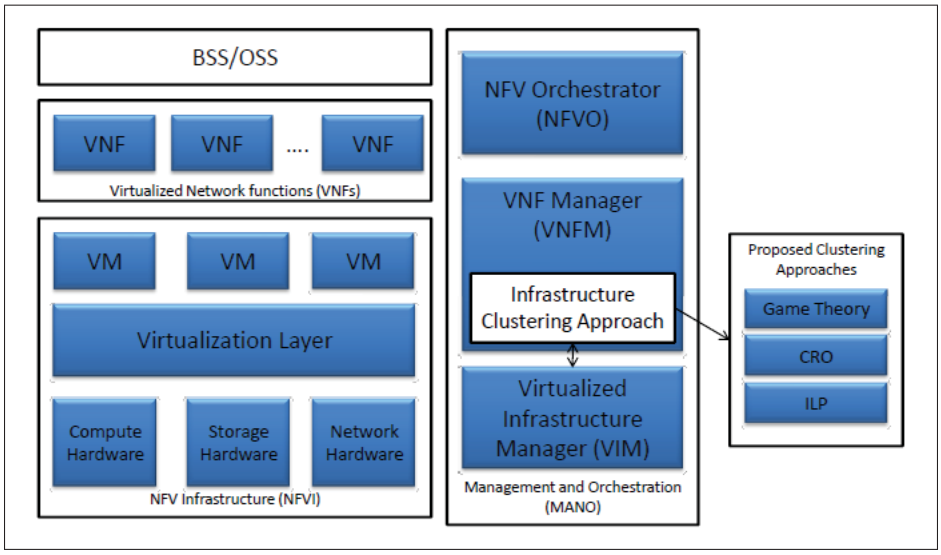


Figure 4.1 Integration of VALKYRIE with respect to MANO architecture

As illustrated in Figure 4.1, the proposed clustering approaches are integrated as a new functionality into VNFM following the NFV MANO architecture ETSI, where 3 different approaches are proposed and compared together, namely, ILP, Game Theory (GT), and Chemical Reaction Optimization (CRO). Another flavor for each the game theory and ILP approaches are introduced. Our proposed approaches interacts with the infrastructure to get all the required information such as the network topology, CPU and memory utilization at any given time to perform the clustering dynamically. Proposing three different approaches will give the

opportunity for the decision makers to choose one of them based on their cluster requirements, they may invoke a CRO-based clustering procedure, a Game Theory-based procedure, or even the ILP solution, to build the desired clusters. This is achieved according to the size of the underlying network topology (small-scale, medium-scale and large-scale networks).

4.5 Related Works

In this section, we review the existing and most relevant approaches that have thus far been proposed in the research literature concerning the clustering of networks for service function chaining.

Although, only very few articles have proposed forming virtual clusters of the physical servers, we will detail the state of the art regarding clustering in cloud environments, especially with respect to virtual machines clustering which aims at optimizing the resource consumption and management of the physical infrastructure. Moreover, we present the techniques adopted for community detection in social networks, which is also an application of clustering in attributed graphs like our considered context.

In Chen, Du, Chen & Wang (2018) propose a topology aware approach to host virtual machines clusters on a compute pool composed of a set of physical machines. The virtual machine cluster is defined as a group of virtual machines hosting a distributed application or service, the challenge here is to place these VMs on top of physical machines in a manner which guarantees that two VMs belonging to the same cluster are not deployed far away from each other to avoid high bandwidth consumption and low performance in terms of the service offered. The authors represent the VM cluster as a directed graph where nodes are the VMs and links between them represent the links. To solve this placement problem, the authors propose a greedy algorithm, which first attempts to host the entire VM cluster on the same physical server if it has enough capacity. Otherwise, it starts with the node having the least connectivity weight and places this VM on the physical machine with the minimum available capacity to host the VM. Once placed, this VM is omitted from the initial graph and the algorithm follows the same procedure

to host the remaining VMs. The authors compare their approach to their previous work in Chen, Chen, Lu & Wang (2017), where the connectivity between VMs was not considered. They also compare it to two basic approaches: the first-come-first-serve and the round-robin Qi *et al.* (2019), Jackson, Bunch & Sigler (2015) approaches which are both used for VM scheduling. The results show that this topology aware technique improves the bandwidth consumption versus the yield with the three other approaches.

Chavan & Kaveri (2014), propose a technique to cluster virtual machines in order to facilitate their scheduling over a shared pool of physical machines, as well as reduce the complexity of their placement, reconfiguration or migration. The goal of this proposed approach is to ensure effective resource sharing between VMs which will provide higher resource availability to end-users. The clustering technique leverages the similarity between VMs in terms of different attributes such as the RAM, the OS or the hardware configuration. K-means was adopted in this paper to perform the clustering. The authors develop also a mathematical model based on linear programming to ensure a better usage of the clustered VMs and enhance their performance.

In U-chupala, Uthayopas, Ichikawa, Date & Abe (2013) propose a technique to create virtual clusters composed of virtual machines dedicated to high performance computing applications. The authors introduce multi-site clustering, since virtual clusters are usually hosted on the same site to avoid performance degradation and keep the QoS offered at a high level. To ensure better management, higher resource availability and shared resources between many end users, the proposed approach focuses more on the connectivity between the VMs over distant sites. An overlay network is used to separate the network of each virtual cluster and ensure that VMs are capable of communicating with one another, even if they are not hosted on the same physical pool of servers. While this approach may offer higher resource availability and greater computational and processing power to clients, it however has a significant effect on the connectivity level: if the same application is distributed across two VMs hosted on different sites, longer delays and and higher bandwidth consumption rates are all but a given.

Abdelsalam, Krishnan & Sandhu (2017) examine the security aspect in IaaS platforms (Infrastructure as a service), where it is highly important to ensure the isolation between the virtual machines serving different clients. The authors employ a modified sequential k-means algorithm-based approach to detect abnormal behavior and anomalies in resource consumption trends across specific virtual machines. Should the algorithm detect abnormal resource consumption peaks, it means the system then may be encountering an external attack from a third malicious party. The clustering is performed on groups of virtual machines based on multiple attributes (e.g. CPU, memory, disk, network throughput.). The k-means approach presented here also takes into account the application's architecture and characteristics. The resulting clusters (web cluster, database cluster, applications cluste, etc.) are grouped based on the nature of the received traffic. A VM is then labeled as malicious if its resource consumption is far from the centroid of its cluster, this labeling decision is based on a threshold fixed by the cloud provider. The algorithm was tested on an Openstack setup and was proven to be efficient in detecting malicious VMs, thus enhancing the security and monitoring over distributed cloud environments and Iaas platforms.

In Wahab, Kara, Edstrom & Lemieux (2019), the authors propose a cluster-based placement approach for virtual network functions. The clustering is considered as a pre-processing phase where the substrate network is divided into a set of coherent groups in terms of specific metrics (Energy, memory, CPU, etc.). The authors used a k-medoids-based approach, and the results proved that this approach helps reduce the overall placement phase length, as well as any needed migration. It is worth mentioning here that the proposed solution takes into account only the servers attributes, while in real case scenarios, we must also consider the delays over the links in the substrate network and their available bandwidth. Moreover, if the clustering is based only on the nodes characteristics it is possible to end up with two servers belonging to the same cluster, with the delays between them being too high, which may in turn increase the SLA/SLO violation percentage in terms of the requested delay and severely degrade the service performance.

Community detection in social networks.

Community detection consists in finding groups of individuals sharing some common characteristics, but also have links with one another that could for example, mean that they are friends

on a social network. In our case, these individuals may be considered as servers with common attributes, and that are linked together in the substrate network.

Liu & Li (2017) study the problem of community detection in the data analysis and processing field. The aim of their work is to improve the quality of clustering over traditional methods, which suffer from high complexity and long execution times. The authors propose to use the genetic algorithm heuristic to form the communities. In this study, the objective function considers only the modularity. While this measure does not take into account the nodes attributes, by simulation, it is shown that the proposed approach manages to obtain acceptable results as compared to the ground truth included in the data-set used. However, no indication is given with regard to the run-time of the algorithm and there was no comparison of the proposed approach to the classical clustering techniques described in the literature.

In Jami & Reddy (2016) compare a set of evolutionary algorithms to form communities in the context of social networks. Three main heuristics are considered: particle swarm optimization, cat swarm optimization and the genetic algorithm combined with simulated annealing. Here again, the authors consider modularity as an objective function. The social network is represented by a weighted graph in which nodes represent individuals and the edges represent the relationships linking them. The comparison of the proposed heuristics is tested on classical data-sets like the Zachary's karate club data-set Zachary (1977), and the results show that the number of communities found by each algorithm, as well as the modularity, is different, but the genetic algorithm approach coupled with simulated annealing exhibits the best trade-off between the quality of the returned solution and also the overall execution time.

Aylani et al. Aylani & Goyal (2017), propose a k-means-based approach to detect communities in social networks. The authors lay emphasis on the interactions between individuals, and take into account their common interests and activities, unlike in classical clustering techniques, where the focus is mainly made on profile similarities (e.g., age, gender, education, etc.). To leverage this aspect of connections between people sharing a social network, the authors introduce a factor called a "common social activity", which considers both similarities in terms of attributes and

of the nature of interactions between individuals. Although, the approach points to a significant shortage in actual clustering techniques, the adoption of k-means also carries major deficiencies, mainly related to the random initial phase, which consists in choosing random first seeds or centroids, and may result in poor solutions. Moreover, k-means do not scale well when the number of attributes considered is high, which is the case in the social networks context.

Many research works employ clustering techniques to form similar groups of items in different fields of application, such as social networks or in cloud computing environments. Most of the works presented above, employ classical approaches based on k-means or k-medoids, for example. Some research papers also adopt evolutionary algorithms, such as genetic algorithm or simulated annealing. In the present paper, we propose clustering algorithms applied to data-centers to form homogeneous groups of servers in terms of selected attributes. This pro-active measure helps reduce the complexity of SFC placement and allows better management of resources on the physical infrastructure, which is important for service providers. To the best of our knowledge, no existing research work has studied topology-aware clustering in the context of a dynamic and proactive approach tailored to NFV.

4.6 Problem Statement, Assumptions and System Model

In this section, we start by providing formal definitions of the physical network infrastructure and listing the terms pertaining to the scope of our work. Then, we present the problem statement before diving into the mathematical model used by the ILP solution, the game theory based approach and the meta-heuristic-based approach.

4.6.1 Definitions and Notations

We now introduce our formal description of the proposed mathematical model, along with the notations used.

Formally, attributed graphs extend the concept of graphs by enriching nodes with a set of attributes. An attributed graph G=(V, E, A) consists of a set of V nodes, a set of links

interconnecting them (E), and the set of node attributes (A) Bothorel *et al.* (2015). Thus, our physical network infrastructure is represented as an attributed graph.

Table 4.1 summarizes the different parameters of our clustering model with the different inputs.

Table 4.1 Notation Table

C	Set of clusters.
C_i	Cluster i.
$Sim(i, j)$	Similarity ratio between servers i and j.
$\delta(C)$	Density function of a partition C.
V	Set of servers.
A	Set of servers attributes.
E	Set of links.
$E(C_i)$	Set of links inside cluster C_i.
$\delta(i, j)$	Equals 1 if servers i and j belong to the same cluster and 0 otherwise.
$A_{i,j}$	The adjacency matrix, $A_{i,j}$ = 1 if servers i and j are directly connected and 0 otherwise.
k_i	The degree of server i.
x_i^d	Vector of attributes associated to the server i.
$\gamma_i^c \in \{0, 1\}$	Equals 1 if server i belongs to cluster c and 0 otherwise.

4.6.2 Problem Statement

We formulate the problem of clustering of attributed graphs in Network Function Virtualization as a Quadratic Assignment Problem, which is recognized in the literature to be one of the most challenging optimization problems. In its formal form, the goal is to assign n facilities to n locations, with the cost of being proportional to the flow between the facilities times the distances between the locations, plus eventually the cost for placing facilities at their respective

locations. Thus, the objective is to allocate each facility to a location such that the total cost is either minimized or maximized, depending on the intrinsic nature of the considered problem.

In our context, the facilities are the servers and the locations represent the clusters we want to form. In summary, we put together facilities that are similar into the same location, while we separate distinct facilities into different locations. The cost functions we use are F1 and F2, which are defined in the next section. As a proof of NP-Hardness, since the Quadratic Assignment Problem is NP-Hard and our model reduces to its form then clustering of attributed graphs in Network Function Virtualization is also NP-Hard Sahni & Gonzalez (1976).

Our goal is to propose a set of clustering techniques tailored to the service function chaining problem, as well as cloud-based virtual network service orchestration using the concept of attributed graphs. The clustering approach attempts to extract non-overlapping clusters using a combined distance that accounts for node features and exploits the characteristics of the underlying networking topologies.

Obtaining a good clustering of an attributed graph requires the optimization of at least two objective functions Bothorel *et al.* (2015) and Zhou *et al.* (2009). There will always be a trade-off between compositional and structural dimensions. These dimensions pertain to the nodes and links, respectively. For node attributed graphs the objectives are :

1) **the structural quality of the clusters**. We consider the modularity function Cheng *et al.* (2011), where a higher modularity corresponds to better clustering. Thus, here we need to maximize this measure as an objective function;
2) **the intra-cluster homogeneity of the node attributes.** Here, we consider the similarity-based measure Bothorel *et al.* (2015), which is the key to building clusters since cluster members are grouped together only when being similar. Thus, we need to maximize this measure (i.e., maximize homogeneity) as an objective function as well.

The justification for these two objectives is purely technical in that these measures are easy to compute and do not require additional information other than that provided by features of the servers and the network topology itself, in terms of number of servers and links. They are thus

not computational time-consuming since their growth is linear. Moreover, as explained in the survey Bothorel *et al.* (2015), for node-attributed graphs, at least two optimization objectives need to be considered. There will be a trade-off between compositional and structural dimensions to be invoked before launching the proposed novel clustering solutions.

4.6.3 Assumptions

An important aspect worth mentioning is the pre-processing task the network topology undergoes. Like any unsupervised learning technique, a.k.a clustering, the data must always be cleaned beforehand, with the links connecting the servers filtered in terms of bandwidth and latency according to the on-demand clusters' requirements. For example, if a link connecting two servers is congested, it will not be included during the clustering process. All the three approaches leverage information (available CPU cores, memory, storage capacity, power consumption, bandwidth and latency) provided by the data plane, which is fetched on a periodic basis using analytic tools such as Grafana, as well as the current network infrastructure topology (the available servers with their inter-connectivity). We assume, without loss of generality, that the network or service provider leverages such mechanisms and tools.

4.6.4 VALKYRIE Clustering Model

4.6.4.1 Objective Functions

Our aim is to build clusters that are similar, while taking into account not only their topological distance, but also the capacity of the links interconnecting them to communicate with each other while within the same clusters.

1) $F1$ stands for the assignment of the servers in the clusters. The aim is to build clusters with similar servers in terms of attributes. Therefore, F1 will maximize the similarity. Yet, clusters with different characteristics may be formed, having either one or multiple attributes in common. For instance, we may use CPU-intensive clusters, energy-efficient

clusters, or even a combination of several attributes to form multi-attribute clusters;

$$\text{F1}=\frac{1}{2}\times\sum_{\forall c\in C,\forall i\in V,\forall j\in V} Sim(i,j) \tag{4.1}$$

where :

$$Sim(i,j)=\gamma_i^c\gamma_j^c[\frac{1}{1+\sqrt{\sum_{\forall d}(x_i^d-x_j^d)^2}}] \tag{4.2}$$

2) $F2$ tries to maximize the modularity for better clustering [6].

$$\text{F2}=\frac{1}{2|E|}\sum_{\forall i\in V,\forall j\in V,\forall c\in C}[A_{ij}-\frac{k_ik_j}{2|E|}]\delta(i,j) \tag{4.3}$$

$\delta(i,j)$ is the Kronecker delta which returns 1 if i and j belong to the same cluster, and 0 otherwise. In our problem formulation, we replace it with the product of γ_j^c and γ_i^c which is equivalent to the Kronecker delta. Thus, F2 becomes:

$$\text{F2}=\frac{1}{2|E|}\sum_{\forall i\in V,\forall j\in V,\forall c\in C}[A_{ij}-\frac{k_ik_j}{2|E|}]\gamma_j^c\gamma_i^c \tag{4.4}$$

4.6.4.2 Constraints

$$\forall c\in C:\sum_{i\in V}\gamma_i^c=1 \tag{4.5}$$

$$\forall i\in V:\sum_{c\in C}\gamma_i^c>=2 \tag{4.6}$$

$$\forall c\in C,\forall i\in V:\gamma_i^c\in 0,1 \tag{4.7}$$

Constraint (4.5) ensures that each server belongs to only one cluster. Thus, the model is outliers-free. Constraint (4.6) ensures that the clusters that are formed have more than two servers inside them. In fact, this parameter can be specified as a different value, depending on the needs of the decision maker or of the network administrator in charge of executing the

model/approaches. Constraint (4.7) specifies that the decision variable is binary (its value is either 1 or 0).

4.7 CRO-based Approach

4.7.1 CRO Approach Description

Chemical reaction optimization is a population-based meta-heuristic that mimics the nature of chemical reactions to solve complex optimization problems. The key component in CRO is the molecule which represents a potential solution of the considered problem. Each molecule in the population is characterized by its potential energy which is the equivalent of the objective functions in the optimization diction. CRO consists of a series of reactions aimed at enhancing the quality of the population over the iterations. These reactions are divided into two primary sets: uni-molecular reactions and multi-molecular reactions.

- **Uni-molecular reactions:** These reactions include only one molecule as an initial input, and may result in a single new molecule, in which case we talk of an On-wall ineffective collision. Alternatively, it may also result in two different solutions. Here, we will be dealing with a decomposition because a single molecule splits into two new molecules having different potential energies.
- **Multi-molecular reactions:** This set consists of two initial molecules, which after the reaction may result in a single molecule, if synthesis occurs, or in two new solutions, if there is an inter-molecular reaction. The synthesis here consists of two molecules colliding with each other and resulting in a different molecule, while with the inter-molecular ineffective collision, two molecules hit each other and then bounce back, and each one of them is slightly changed, but independently from the other.

Both types of ineffective collisions allow intensifying the population by performing a local search in the solution space, while decomposition and synthesis act more as diversification procedures. We thus have a balanced number of operations that includes all these reactions, thereby ensuring an effective search in the solution space and allowing near-optimal solutions to

be found for the considered optimization problem Lam & Li (2012). The overall CRO approach is driven by the following parameters:

- **Potential Energy:** This is a quality measure that measures the objective function of a given solution/molecule.
- **Kinetic Energy KE:** This measure quantifies the tolerance of the whole system to accepting solutions that are worse than the initial ones.
- **Decomposition rate A:** At each iteration, this value determines the uni-molecular reactions to be applied: decomposition or on-wall ineffective collision.
- **Synthesis rate B:** At each iteration, this value determines the multi-molecular reaction to be applied: synthesis or inter-molecular ineffective collision.
- **Maximum number of iterations:** This is a counter that is used as a stopping criterion. Once the maximum number of iterations is attained, the CRO heuristic converges to the most optimal solution available in the current population.

These values are set by the network administrators as inputs and should be tuned to obtain the best optimal solution depending on the optimization problem considered. In the following section, we will describe how we used the chemical reaction optimization meta-heuristic for our clustering problem. We also define the molecule encoding as well as the operators we used for each of the elementary reactions.

4.7.2 Molecule Encoding

One possible way to encode the solution is to use the grouping technique as defined in El Mensoum, Wahab, Kara & Edstrom (2020) where each cluster will represent the group of servers it contains as described in Table 4.2.

Table 4.2 First molecule encoding

Cluster ID	1	2	3	4
Included nodes	1,3,7	2,4,6	5,8,	9,10

The table 4.2 shows a topology of 10 nodes distributed on three different clusters. Using this encoding, if we apply the usual operator used for the chemical reaction optimization or genetic algorithm we may end up with overlapping clusters which will violate the constraint (4.5) which requires that each node should belong to only one cluster. To avoid this problem we decided to go with the encoding shown in Table 4.3.

Table 4.3 Adopted molecule encoding

Server ID	1	2	3	**4**	5	6	7	8	9	10
Cluster ID	1	2	1	**2**	3	2	1	3	4	4

This encoding consists of an N size array where N is the number of nodes in the substrate graph; each server is assigned to a selected cluster which has a specific label.

4.7.3 Initial Population

Our CRO approach consists of a centroid-based clustering where we specify a set of points in the data-set and then assign the rest of our servers to one of the defined centroids based on the similarity measure we defined in section 4.6.2. Prior to executing the different CRO operators, the choice of the initial k centroids is a critical step. The challenge in this phase consists in ensuring that these centroids are spread out enough to reflect the distribution of our servers in the data-set in order to allow us to distinguish the different clusters/groups that we have in the studied network topology. Inspired by k-means++, we decided to use the variance of our considered servers in terms of their attributes and define the initial centroids accordingly. This ensures that the initialization phase would not be totally blind or random, and that the centroids chosen would be representative of the variety of our servers, which would then allow us to enhance the quality of our initial population members: that is a key part of every meta-heuristic approach. The step-wise process to generate the molecules is described in Algorithm 4.1.

Algorithm 4.1 Molecule Generation

Input: Number of clusters ***K***, Servers attributes, Population size ***Popsize***
Output: Initial population
1: Initialize counter $count = 0$
2: Generate a random vector $V[0]$ of K points in the range of servers' attributes
3: **while** $count \neq Popsize$ **do**
4: Assign each data point in the space to the closest centroid in vector $V[count]$
5: Add the formed molecule to population
6: increment $count$, i.e., $count++$
7: Create a new vector $V[count]$=$V[count--]$+$dataset\ variance$; {this step allows to initialize new centroids shifted from the previous ones using the data-set variance.}
8: **end while**
9: **return** Initial population

4.7.4 Elementary Reactions

In this section, we will describe how we used each of the CRO operators to enhance the quality of the formed solutions present in the initial population. As we previously explained in section 4.7.1 the operators of CRO consist of two major sets:

- **Uni-molecular reactions:** The first reaction in this set is called the *On-wall ineffective collision*. This operator consists of a single input molecule representing a potential solution called *M1* in Algorithm 4.2 and then results in a new solution that we call *Mo*. We select a set of nodes that have low similarities within their assigned clusters, re-compute their similarities to the updated centroids and then re-assign them to the cluster to which they are the most similar in terms of the selected attributes (steps 1 to 3 in Algorithm 4.2). This way will allow us to correct the misplaced nodes and re-enforce the homogeneity of the formed clusters. Once the new solution is formed we compute its modularity. If it is higher than the initial input solution M1, we destroy this later and add Mo to the population; otherwise; we keep M1 and destroy the resulting Mo.

The second operator in this category is the *Decomposition*. This reaction also starts with a single molecule M1 but results in two distinct solutions D1 and D2. We select a random node in the initial solution M1 which ranges between 0 to N where N is the number of nodes in our data-set.

Algorithm 4.2 On-wall ineffective collision

Input: One initial molecule M1 , Servers attributes
1: Select the set of nodes with low similarity to their clusters
2: Compute the similarity of these nodes to the centroids of M1
3: Re-assign each node to its closest centroid based on similarity
4: Compute the Modularity of the newly formed solution Mo
5: **If** Modularity(Mo) > Modularity(M1)
6: Destroy M1
7: Add Mo to the population
8: **else**
9: Destroy Mo
10: **end If**

We keep the first part of this molecule M1 in the first resulting solution D1 meaning we copy the clustering from Server[0] to Server[R] (step 2 Algorithm 4.3) and then assign the rest of nodes to their closest centroids based on similarity(step 3 Algorithm 4.3). The second solution is constructed from the second part of M1 but this time we keep nodes from Server[R+1] to Server [N] (step 4 algorithm 4.3) and assign servers 0 to R to their closest cluster again based on similarity (step 5 Algorithm 4.3). Once the new offspring is formed we compute the modularity of both D1 and D2 if one of them is higher than that of the initial solution M1 we destroy this later while adding D1 and D2 to the population; otherwise; the new solutions are destroyed and the population remains unchanged.

- **Multi-molecular reactions:** The first operator in this category is the Synthesis, and its full process is described in Algorithm 4.4. We have as an input two molecules representing two potential solutions chosen randomly from the current population M1 and M2, the collision between these two elements results in a single output molecule Ms which is also a probable solution of our optimization problem. This reaction consists of choosing a random point on both parents M1 and M2 and populating the offspring solution with elements from the 1st parent ranging from server[0] to server[R] and then the rest of Ms is formed by the 2nd parent ranging from server[R+1] to server[N] where N is the number of servers considered in our data-set (step 2 in Algorithm 4.4). If the resulting modularity of the new formed solution Ms is higher than that of the initial molecules M1 and M2 then these two are later discarded

Algorithm 4.3 Decomposition

Input: One initial molecule M1 , Servers attributes, Number of servers *N*
1: Select a random integer R ranging from 0 to N
2: Copy M1[0:R] in D1[0:R]
3: Assign nodes from R+1 to N to their closest centroids based on similarity to form D1[R+1:N]
4: Copy M1[R+1:N] in D2[R+1:N]
5: Assign nodes from 0 to R to their closest centroids based on similarity to form D2[0:R]
6: Compute the modularity of D1 and D2
7: **If** Modularity(D1) > Modularity(M1) **OR** Modularity(D2) > Modularity(M1)
8: Destroy M1
9: Add D1 and D2 to the population
10: **else**
11: Destroy D1 and D2
12: **end If**

while Ms is added to the population, otherwise, the population remains the same and both M1 and M2 are kept in the population, while Ms is destroyed (steps 4 to 8 in Algorithm 4.4).

Algorithm 4.4 Synthesis

Input: Two initial molecule M1 and M2 , Number of servers
1: Define a random integer R ranging from 0 to N
2: Construct the new solution Ms= M1[1:R] *U* M2[R+1:N]
3: Compute the modularity of Ms
4: **If** Modularity(Ms) > Modularity(M1) **OR** Modularity(Ms) > Modularity(M2)
5: Destroy M1 and M2
6: Add Ms to the population
7: **else**
8: Destroy Ms
9: **end If**

The second operator in this category is the Inter molecular ineffective collision. This reaction also starts with two molecules M1 and M2 and results in two potential solutions C1 and C2. In this reaction each new molecule is generated from its parent independently from the second one, so we use the on-wall ineffective collision defined in Algorithm 4.2 in order to generate two distinct solutions: ***C1=ON-WALL COLLISION (M1) and C2=ON-WALL COLLISION (M2)***.

4.7.5 Overall CRO-based Clustering Algorithm

In Algorithm 4.5 we describe the step-wise methodology to execute the overall steps of the CRO-based clustering approach process. The first step consists in generating a set of feasible solutions that is equal to the population size which we define as an input using Algorithm 4.1. We start first by generating a random number **B** between [0,1] (step 4 Algorithm 4.5) if **B** is higher than **Mol** then we randomly select one Molecule from the initial population generated by 4.1. The next step is to verify the number of hits attained if it is higher than the value of **A** (specified as an input to the Algorithm 4.5), then we apply the decomposition process 4.3. Otherwise, the on-wall ineffective collision process (Algorithm 4.2) will be executed. The other scenario is for the case where the generated number **B** is lower than the input **Mol**. Here, we apply the multi-molecular operators. We first check if the set kinetic energy is lower than the value of **B** (specified as an input to the Algorithm 4.5), then the synthesis operator (Algorithm 4.4) is executed. Otherwise, we apply the inter-molecular ineffective collision process and we decrements the kinetic energy. This process of CRO operators will be repeated as long as the total number of iterations has not been attained, which is the stopping criterion for this algorithm. Once the population reaches the maximum number of iterations set as an input, the optimal solution is chosen with respect to modularity as defined in section 4.6.4. The solution that returns the highest modularity is selected as the optimal solution.

4.8 Game Theory-based Approach

In this approach, the Stable Roommate (SR) algorithm was used to find what are known as stable matching pairs between servers,such that no two non-matched servers prefer each other more than their actual matching. This algorithm comprises two phases: the preference list computation phase and the preference cycle elimination phase. In the first phase, the attributes of each server were given, along with the list of servers in the topology. The preference table was then built by calculating the similarity between each server *i* and every other server *j* in the substrate network. The similarity values calculated were ranked in descending order, and formed the preference table as shown in Algorithm 4.6. After the preference table was built, each server

Algorithm 4.5 Overall CRO Algorithm

Input: Number of servers, Servers attributes, Population size, MaxIteration, A=0.6*MaxIteration, B=0.3*MaxIteration, KineticEnergy=MaxIteration, Num-Hits=0
Output: Final Solution ***O***

```
Generate randomly an number Mol in the range [0,1]
Generate initial molecules by executing Algorithm 4.1
while Num-Hits < MaxIteration do
    Generate randomly B in the range [0,1]
   If B > Mol do
        Randomly select one molecule
      If Num-hits > A do
       Execute Decomposition Algorithm 4.3
      else
       Execute On-wall ineffective collision Algorithm 4.2
      end If
   else
    Randomly select two molecules from the population
      If KE < B do
       Execute Synthesis Algorithm 4.4
      else
       Execute Inter-molecular ineffective collision
      end If
       KineticEnergy–
   end If
Num-Hits ++
end while
Check for a new optimal solution returning the highest modularity
return Final solution O and its modularity
```

Algorithm 4.6 Preference List Computation

Input: Set of attributes for all servers ***A***, list of servers ***V***
Output: Preference List

```
foreach server i ∈ V
    foreach server j ∈ V
        Calculate the similarity between servers i and j and store them in sim array.
        Sort sim array in descending order to form 2-dimensional sim array.
    end
end
return Preference List
```

should be uniquely paired with only one other server in the substrate, which is accomplished by applying the Irving Algorithm Irving (1985). The inputs of this algorithm are the preference table, along with the list of servers in the substrate topology, with their respective attributes, as briefly described in Algorithm 4.7. The Irving algorithm matches each server to only one other server. Each server i sends its proposal to the most preferable server j as computed in the previous algorithm and specified in the preference table. If any server has more than one proposal at a time, it will keep the best one and discard the others. This process is repeated until each server has only one proposal. Based on the position of the current proposal for each server in the preference table, all the servers that are less preferable will then be discarded from the preference table. This generates a shortlisted preference table, such as the one in figure 4.2. The last step in the Irving algorithm is to eliminate all the preference cycles existing in the shortlisted preference table. This process is performed by checking all the rows in the shortlisted

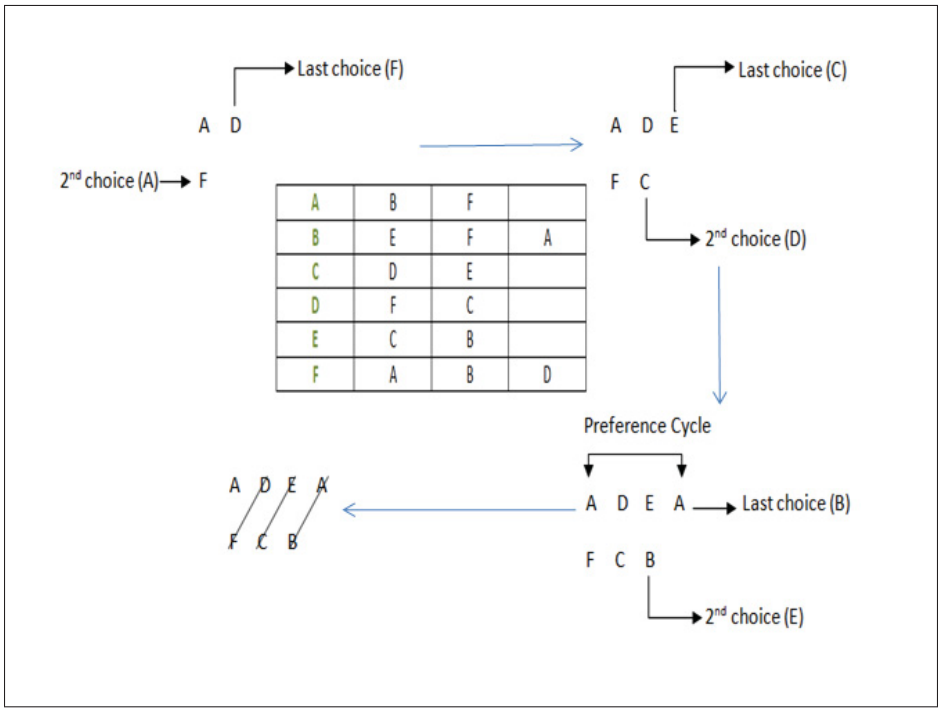


Figure 4.2 Preference Cycle Elimination Process

preference table, which has more than one preference. For instance, as shown in Figure 4.2, server A has two preferences, namely, are B and F (B is more preferable). In order to detect

the existing preference cycles, starting from the first row, the second_choice(server A) and the last_choice(second_choice(server A)) are noted, and this step is repeated until the preference cycle is detected, as shown in the figure. Once this cycle is detected, the following preferences between (A and B),(E and C) and (D and F) are discarded. This process is repeated until each server has only one preference in the table, which is called the stable matching solution. The

Algorithm 4.7 Irving Algorithm

Input: The servers set V and their attributes and the preference list.
Output: The stable matching.
1: Matching-pairs = Irving(V, $PreferenceList$)
2: **return** Matching-pairs

initial clusters are formed by pushing portions of the generated pairs into different clusters based on similarity characteristics. This clustering is improved by checking if adding one of the servers to a specific cluster would improve the modularity value for the whole partition. If this is the case, the server is added to one of the clusters. This process is performed iteratively in order to maximize the modularity value for the whole partition while minimally affecting the similarity value for each formed cluster, as shown in Algorithm 4.8.

The centroid value of each cluster is updated dynamically at each time the server is added or removed from the cluster. Two flavors of the game theory approach are proposed in this paper, one of which focuses more on maximizing the modularity, with the existence of outliers, while the other one has no outliers with a reduced value of modularity. If Algorithm 4.8 generates clusters with outliers and the cloud provider prefers to have no outliers, then our solution will force each server that does not belong to any cluster to join one of the clusters based on the similarity value between this server and the centroid of this cluster.

4.9 Asymptotic Analysis

We will now describe the algorithmic complexity of our proposed suite of VALKYRIE approaches. It is worth mentioning this complexity analysis is performed for the worst-case scenario. It is clear that the way our mathematical model is described is NP-Hard since it

Algorithm 4.8 Clusters Improvement and Forming

```
Input: Matching pairs, Clusters
Output: Clusters of servers
foreach Cluster ∈ Clusters
    foreach pair ∈ matchingpairs
        Check the similarity between each server in pair and the centroid of the cluster
        if(Similarity exists) then
            if(already in cluster) then
                keep the sever inside the cluster.
            else
                if(adding this pair to the cluster maximize modularity) then
                    join the cluster.
                    remove the pair from the matching pairs.
                else
                    do not join the cluster.
                    check the next pair from the matching pairs.
                end if
            end if
        end if
    end for
end for
return clusters of servers
```

embodies the form of the well known Quadratic Assignment Problem Sahni & Gonzalez (1976).

CRO-based approach. Our CRO-based approach consists of four algorithms, as described in section 6. We detail the complexity of each one as follows:

- Algorithm 4.1 runs in $O(Popsize \times V)$
- Algorithm 4.2 runs in $O(V^2 + V)$
- Algorithm 4.3 also runs in $O(V^2 + V)$
- Algorithm 4.4 runs in $O(V^2)$

Based on this analysis, Algorithm 4.5 (CRO-based approach) complexity is $O(MaxIteration \times (V^2 + V))$

Game theory-based approach. Similarly, three algorithms were employed and their complexities are computed as follows:

- Algorithm 4.6 runs in $O(V^3 \times Log(V))$ as we use TimSort algorithm to sort the similarity array which runs in $O(V \times Log(V))$ Cormen,Thomas H and Leiserson (2009)
- Algorithm 4.7 runs in $O(V^2)$
- Algorithm 4.8 runs in $O(V^3)$

Based on this analysis, the Game Theory-based approach complexity is $O(V^2 + V^3 \times Log(V))$

4.10 Evaluation

In this section, we evaluate VALKYRIE. We assess our chemical reaction optimization algorithm both on small and large-scale networks. We evaluate the quality and effectiveness of the solution with and without the presence of ground truth.

4.10.1 Setup

We implemented the mathematical model using Python 2.7 and solved it thanks to Gurobi 7.5.1 to get the optimal solution. Our chemical reaction optimization and game theory procedures were implemented using Python 3.7. The experiments and implementations were carried out on a physical machine composed of 8 CPU cores.

Two types of network infrastructures were considered to evaluate VALKYRIE. We generated the topologies using the NetworkX library. We report the average values from the experiments which were repeated 10 times.

4.10.2 Performance Metrics

We evaluate VALKYRIE according to the following metrics to assess its effectiveness :

1) **Runtime** : we calculate the time taken by the different approaches to partition the network into the number of desired clusters;
2) **Similarity** : we evaluate the average similarity of all the clusters;
3) **Modularity** : we evaluate how dense the connections between the nodes are within the clusters and how sparse they are while in different clusters;
4) **Density** : we compute the proportion of edges that lie within the clusters, and a higher density corresponds to a better clustering. It is defined as follows:

$$\delta(C) = \frac{1}{|E|} \sum_{\forall C_i \in C,} |E(C_i)|$$

where $E(C_i)$ is the set of edges that are inside the ith cluster.
5) **Outliers** : we compute the number of servers that do not belong to any cluster.

It is worth mentioning that $Density$ is the validation technique we used to assess our clustering approaches. Yet, the values of all the metrics, shown in the y-axis of the figures, are in the range [0, 1] and the higher the value the better it is.

4.10.3 Scenarios

Our approaches are evaluated under a set of scenarios, which we define here-under. It is worth mentioning that we have designed two variants of the game theory based approach. Game Theory (GT) based approach makes use of algorithms 4.6, 4.7 and 4.8 while Enhanced-Game Theory (E-GT) based approach improves GT that suffers from outliers.

4.10.3.1 Scenario 1

In this scenario, we assume that the topology is modular, and we consider it as the ground truth. In such a scenario, the distribution of CPU cores per module is defined as follows : 4-11 for the first module, 12-24 for the second one, and 25-36 for the third one.

4.10.3.2 Scenario 2

In this scenario, we assume different topologies in terms of the number of servers with fixed connectivity degree for each server. The servers in this topology are randomly connected with each other, and they are not modular, as in the case with the previous scenario. The distribution of CPU cores is drawn between 4 and 64 units.

Table 4.4 Degrees of connectivity in our scenarios

Network size	Scenario 1	Scenario 2
20	2	3
50	7	3
100	14	3
200	29	3
300	44	3
500	74	3
1000	149	3

4.11 Discussions and Observations

In this section, an analysis of the results is presented for three different topologies to test our approaches. It is worth mentioning that for all the evaluations, regarding ILP, we do not report the results for network sizes more than 200 as the runtime is exponentially increasing and takes several hours. Instead, we devised a sub optimal version of ILP which we denote by ILP_subopt by tuning the optimality gap parameter as defined in Gurobi to obtain quick solutions.

4.11.1 Modular Topology with Variable Connectivity Degree

Based on figure 4.3, it can be seen that the clustering time for the ILP increases exponentially as the network size increases. For example, it takes hours once the number of servers exceed 200. This behavior is expected as the ILP attempts to find the exact solution that justifies the adoption of one of the developed heuristics. With regard to the other heuristics approaches that were

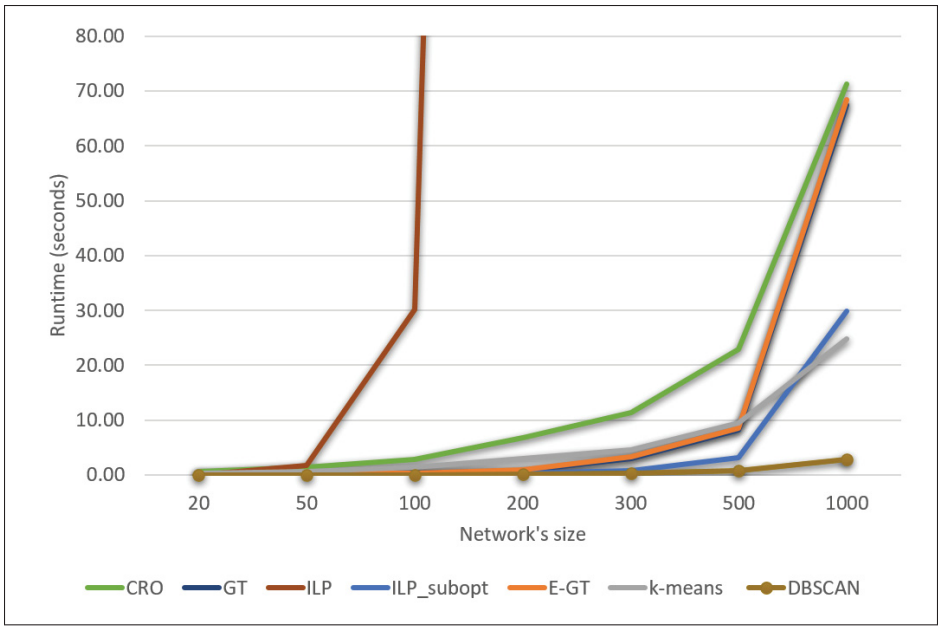


Figure 4.3 Runtime

considered, it can be seen that CRO increases from 0.6 seconds for 20 servers to 71 seconds for a topology of 1000 servers. For the two flavors of Game Theory approaches, the value of runtime ranges from 0.2 to 67seconds for GT and from 0.03 to 68 seconds for E-GT, While for ILP_subopt, it ranges from 0.11 to 29 seconds. Although ILP_subopt has the minimum runtime, all the heuristics approaches are still in an acceptable range to be considered as a clustering solution by a cloud provider since clustering is done on a periodic basis and in a proactive way. DBSCAN and K-means clustering algorithms are mostly giving less runtime, but as mentioned in the asymptotic analysis, the running time for our proposed approaches is fairly acceptable.

Another metric that requires evaluation is the similarity, as shown in Figure 4.4, the ILP identifies the best value in terms of similarity compared to the other approaches for a small-scale environment (up to 200 servers).When CRO and the two flavors of GT heuristics are compared, it can be seen that the values are close, with GT and E-GT demonstrating slightly better results in terms of similarity. This may be attributed to the fact that the GT approach has outliers and although E-GT has slightly better similarity values, it compares unfavorably to CRO in terms of modularity. With regard to the Game Theory approach, not all servers are included in the

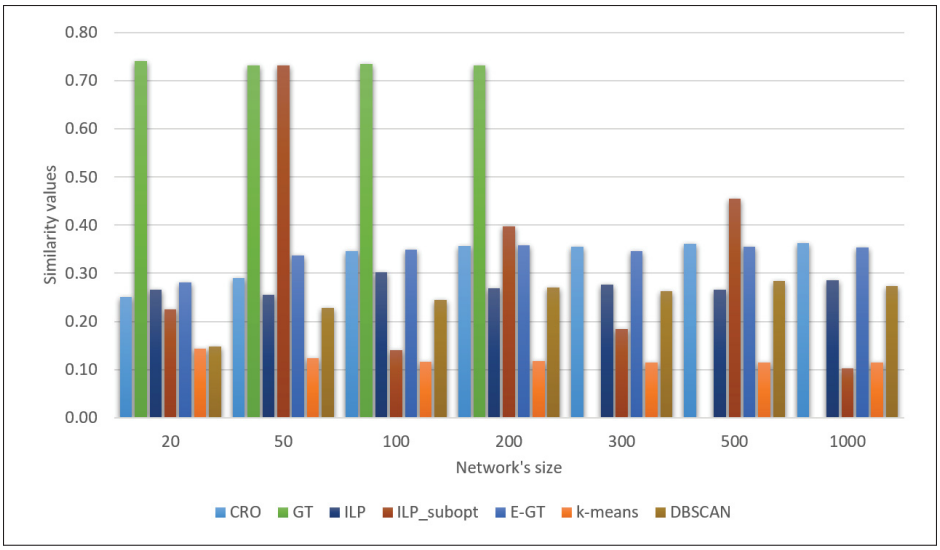


Figure 4.4 Similarity

formed clusters, which explains the difference in similarity, albeit that this difference is fairly small at an average of 0.05. When comparing our proposed algorithms with DBSCAN and K-means, the results reveal that our proposed approaches have better similarity values most of the time regardless of the network's size.

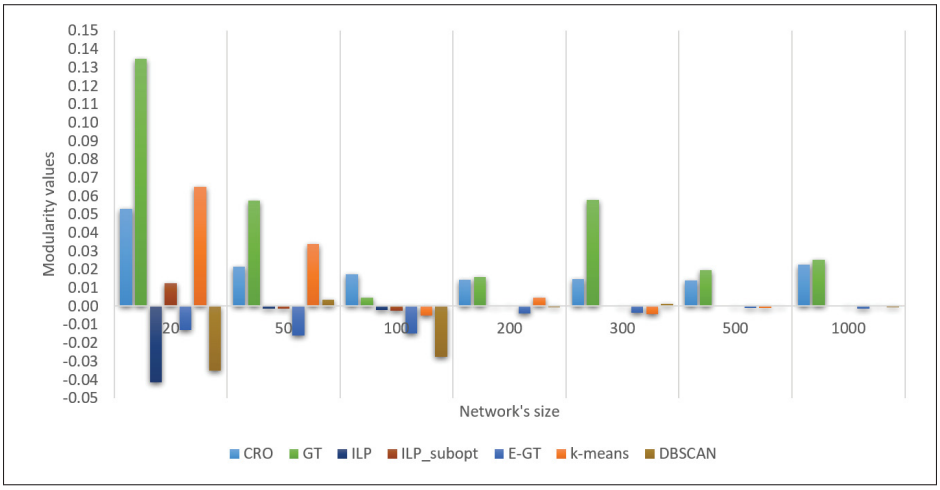


Figure 4.5 Modularity

Figure 4.5 shows the differences between the considered approaches in terms of modularity. The ILP and ILP_subopt solutions find the lowest values for all type of topologies compared

to CRO and the two flavors of GT, which is mainly due to the simplex algorithm that finds the optimal tradeoff between both considered objective functions. As shown in figure 4.4, ILP finds the best similarity but figure 4.5 illustrates that this has an impact on the modularity. As for the heuristics CRO and the two flavors of GT, it can be seen that GT finds better results for small topologies ranging from 20 to 50 servers and CRO takes the lead for medium topologies of 100 servers; the latter have quite similar values for large topologies of more than 200 servers, with an advantage to the GT approach. The difference in modularity can be attributed to the outliers that GT approach returns. Another takeaway from this experiment is that our proposed approaches display better modularity values when compared with DBSCAN and K-means regardless of the network's size.

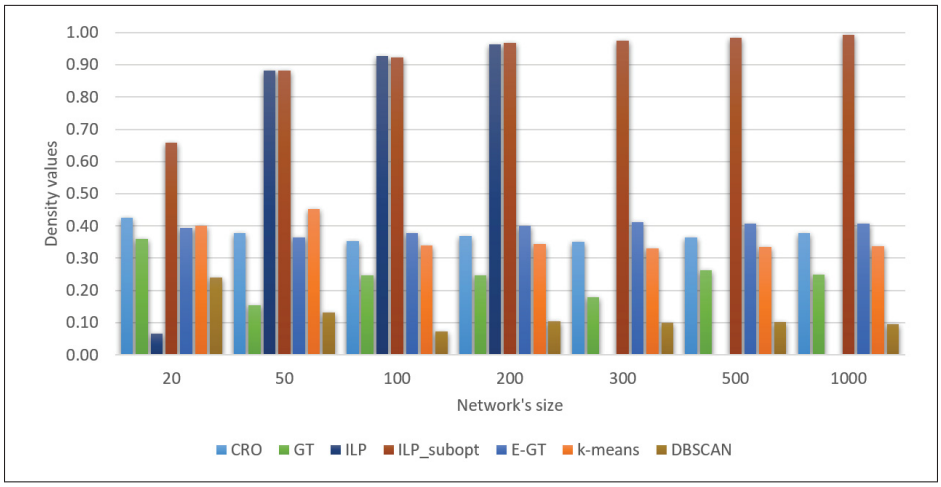


Figure 4.6 Density

The density is an internal validation measure that is used to evaluate the quality of the clustering when the ground truth is not known in advance, meaning that the clusters are not known before the algorithms are applied. Based on Figure 4.6, it can be seen that CRO and E-GT have almost the same density values which are better than the other approaches. This is mainly because they do not return any outliers. The clusters formed by CRO and E-GT include more servers and are more dense compared to those returned by GT, where the number of outliers jumps up to 25% of the substrate for large topologies composed of 1000 servers. Although ILP-SubOpt exhibits the highest density, it is not deployable since some clusters contain only two servers, which is

not a preferable clusters for the network administrator to have. The density values obtained by our proposed approaches are mostly better regardless of the network's size than those obtained by the traditional DBSCAN and K-means clustering techniques.

4.11.2 Random Topology with Fixed Connectivity Degree

Like the previously tested topologies, the ILP solution takes a considerable amount of time to converge to the optimal solution as shown in figure 4.7, and increases sharply as the network size grows. The two Game Theory flavors approaches take less time to converge to the sub-optimal compared to the CRO and ILP_SubOpt approaches and all of them except the ILP_SubOpt solution have a lower runtime compared to the previous topology, where the degree of each node varies. This is an expected result because the substrate network is less complex when the degree is fixed compared to varying connectivity degree , as the latter not only means a higher number of edges and more computational time but also a greater cost to compute the values of modularity. DBSCAN and K-means techniques exhibit a slightly better running time compared to the other approaches in particular for DBSCAN, but our proposed approaches still have a fairly acceptable running time.

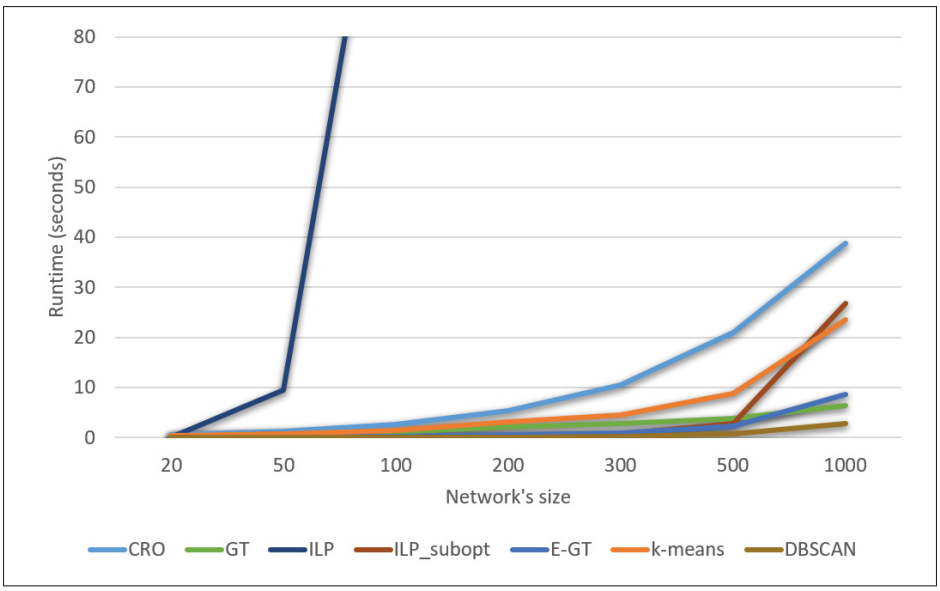


Figure 4.7 Runtime

There is no major difference between the behavior of CRO, GT and E-GT approaches when the degree is fixed because this measure is more related to the servers' attributes rather than the connectivity between them. GT finds a slightly better results compared to CRO when it comes to the similarity as shown in figure 4.8, but again with an existence of outliers which explains this difference. ILP and ILP_subopt solutions exhibits the highest values of similarity for the small-scale environment (up to 200 servers) and finally GT and E-GT have close values to each other. The results have also shown that some of our proposed approaches outperform the results obtained by DBSCAN and K-means.

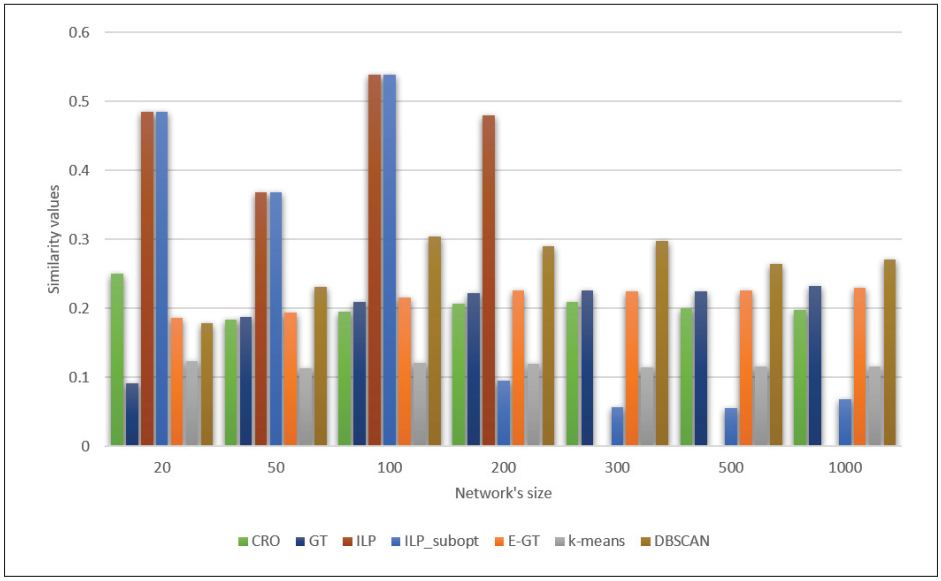


Figure 4.8 Similarity

The difference of modularity is higher in this topology between GT and all the other approaches as shown in figure 4.9. We can see that GT find better results for topologies composed of above 20 servers, but the number of outliers also jumps in this topology which explains again this gap. The ILP and ILP_subopt solutions have the same behavior compared to other the other topology. Yet, in this setting, the modularity values for our proposed approaches are mostly better compared to the DBSCAN and K-means.

For this type of topologies CRO is the best in terms of density, as shown in figure 4.10, compared to GT and E-GT. However, E-GT has better values compared to GT since it does not have any

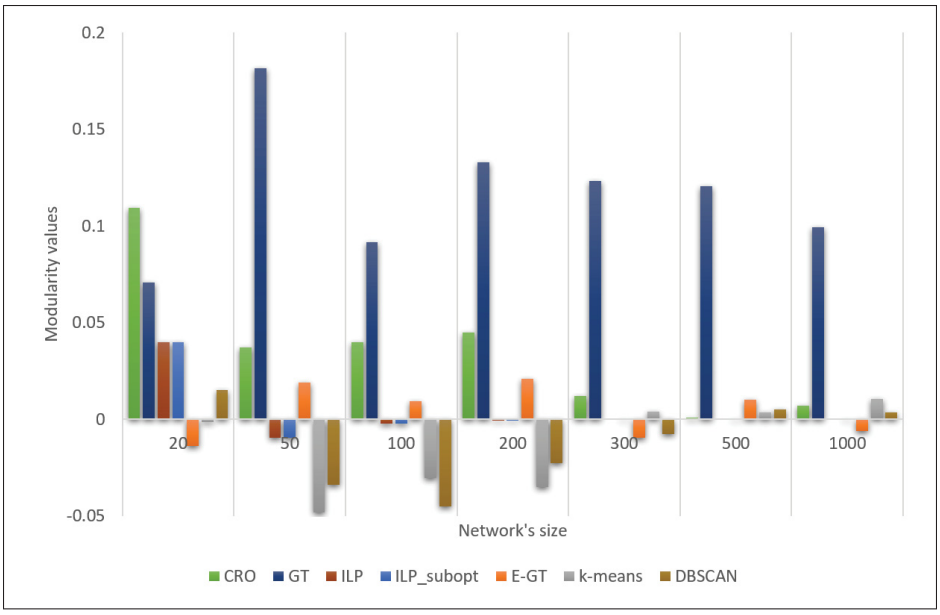


Figure 4.9 Modularity

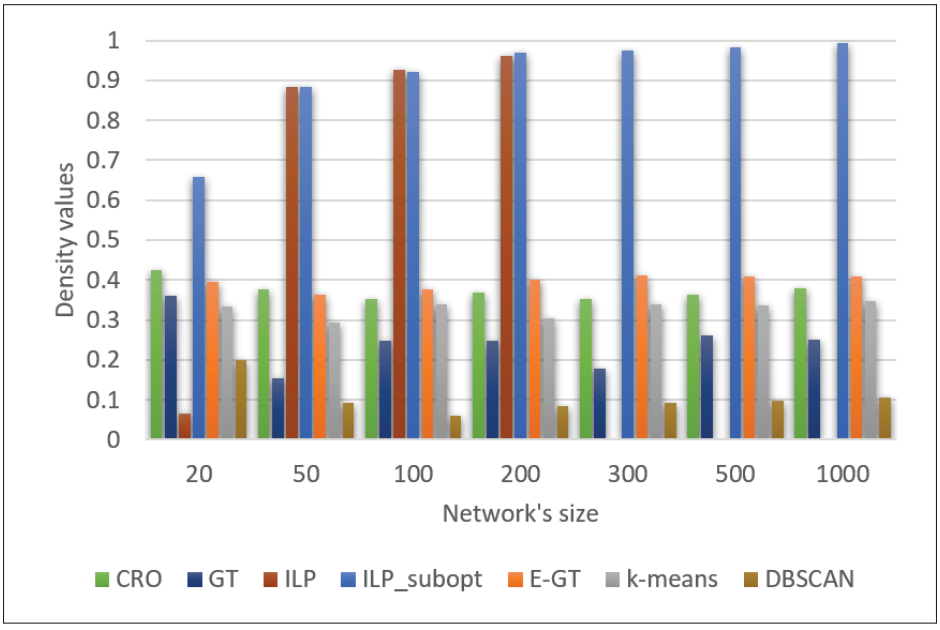


Figure 4.10 Density

outliers and the number of edges inside each formed cluster is higher. ILP_subopt solution shows the best value in terms of density, although it puts the servers among the clusters unevenly.

In fact, one cluster may be formed only with two servers which is not ideal and appropriate for a cloud provider. Again our proposed approaches have shown better density values compared to the traditional DBSCAN and K-means algorithms.

As shown in table 4.5, the number of outliers increases in a linear manner as the network size increases for both topologies. This only occurred in one of the proposed approaches which is the GT approach, as all of the other approaches are respecting the constraints defined in the mathematical model. Having some outliers had an impact on getting better values for similarity (figure 4.8) and modularity (figure 4.9), which could benefit a cloud provider since there could be certain cases where the Service Function Chain (SFC) request has a specific requirement and preference for a cluster to be deployed with higher modularity and similarity values. On the other hand, there are some outliers that were detected in both topologies when applying DBSCAN algorithm as shown in Table 4.6.

Table 4.5 Number of outliers using Game Theory

Network's size	GT (scenario 1)	GT (Scenario 2)
20	7	11
50	23	19
100	22	29
200	45	76
300	113	139
500	104	207
1000	242	454

Table 4.6 Number of outliers using DBSCAN

Network's size	DBSCAN (scenario 1)	DBSCAN (Scenario 2)
20	5	3
50	1	3
100	1	0
200	0	1
300	0	0
500	0	0
1000	0	0

Based on the presented results we can draw the following conclusions:

- The clustering time is influenced by the number of servers, the number of links and the connectivity degree of each server.
- Game Theory based approach is better in terms of modularity compared to other proposed approaches,but it suffers from outliers. It is calculated only with the clustered servers as the others are outliers.
- ILP solution has the best value of similarity for small scale environment ($\leq$ 200 servers), whereas GT has a slightly better values compared to other heuristics but with the existence of outliers.
- If cloud provider prefers no outliers when performing the substrate clustering, then CRO is best heuristic solution to choose since it has a close similarity values compared to other heuristics and with the highest values of modularity in all kind of topologies. Although ILP_subopt might have a higher similarity values compared to CRO, but the servers are not evenly distributed between clusters and you might have two servers inside some clusters.
- Cloud provider could choose to perform clustering using ILP_subopt solution if clusters with high similarity values are needed, which could be the case in some scenarios.
- Density shows that CRO is better compared to the two flavors of Game Theory approach only in random topologies but not in the modular one, where E-GT has a higher value. Although CRO has less density value compared to ILP_subopt solution in all kinds of topologies but in most of the cases it can be preferable to be chosen by the cloud provider to have clusters which are almost having even number of servers inside them.
- ILP is better for small scale networking topologies. However, for large scale networks CRO and Game Theory are a better choice because they exhibit reasonable clustering runtime and a good tradeoff between similarity and modularity.
- Overall, our proposed approaches exhibit, most of the time, better performances compared to K-means and DBSCAN in terms of similarity, modularity and density but with an acceptable additional runtime.

4.12 Conclusion

In this paper, we presented VALKYRIE, a suite of solutions for the clustering and partitioning of large attributed graphs for virtualized and non-virtualized environments. This was in a bid to help decision makers get rid of scalability and computational time burdens when deploying their services (cloud and non-cloud) in cases where each end user's requirements could be completely different from those of others. We first defined the system architecture, formalized the problem, and then we presented the system model. The problem was formulated using a Mixed-Integer Linear Program to get the optimal solution for small scale sizes, while a chemical reaction-based meta-heuristic and a game theory-based approach were proposed to handle the scalability issues in the mathematical program. Our approaches consider the attributes on the nodes and the network jointly by optimizing the similarity and modularity, respectively, as cost functions. Experiments with different infrastructures have shown that our solutions achieve reasonable clustering time with respect to their size and are able to find a good trade-off between density, modularity, similarity as a set of cost functions, in addition to being outliers-free. Yet, our proposed approaches were also compared with the two well-known clustering techniques, namely DBSCAN and K-means and results have revealed a better performance in both topologies. One of the proposed solutions is a potential candidate to be chosen by the cloud/service provider, as it lends them more flexibility, depending on the desired use case and the requirements of the service function chains in the context of NFV.

Finally, given their low computational complexities, our techniques are viable solutions to be integrated into orchestration systems following the NFV MANO framework.

Acknowledgments

This work has been supported by Ericsson Canada and the Natural Sciences and Engineering Research Council of Canada (NSERC). The authors would like to thank Mohssine Arrouch, a Master's student at School of Superior Technology(ETS), University of Quebec (Canada), for providing us with the experimental results using DBSCAN and K-means.

CHAPTER 5

DAVINCI : ONLINE AND DYNAMIC ADAPTATION OF EVOLVABLE VIRTUAL NETWORK SERVICES OVER CLOUD INFRASTRUCTURES

Laaziz Lahlou[1] , Nadjia Kara[1] , Claes Edstrom[2]

[1] Département de Génie Mécanique, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3

[2] Ericsson, Canada,
8275 Rte Transcanadienne, Saint-Laurent, QC H4S 0B6

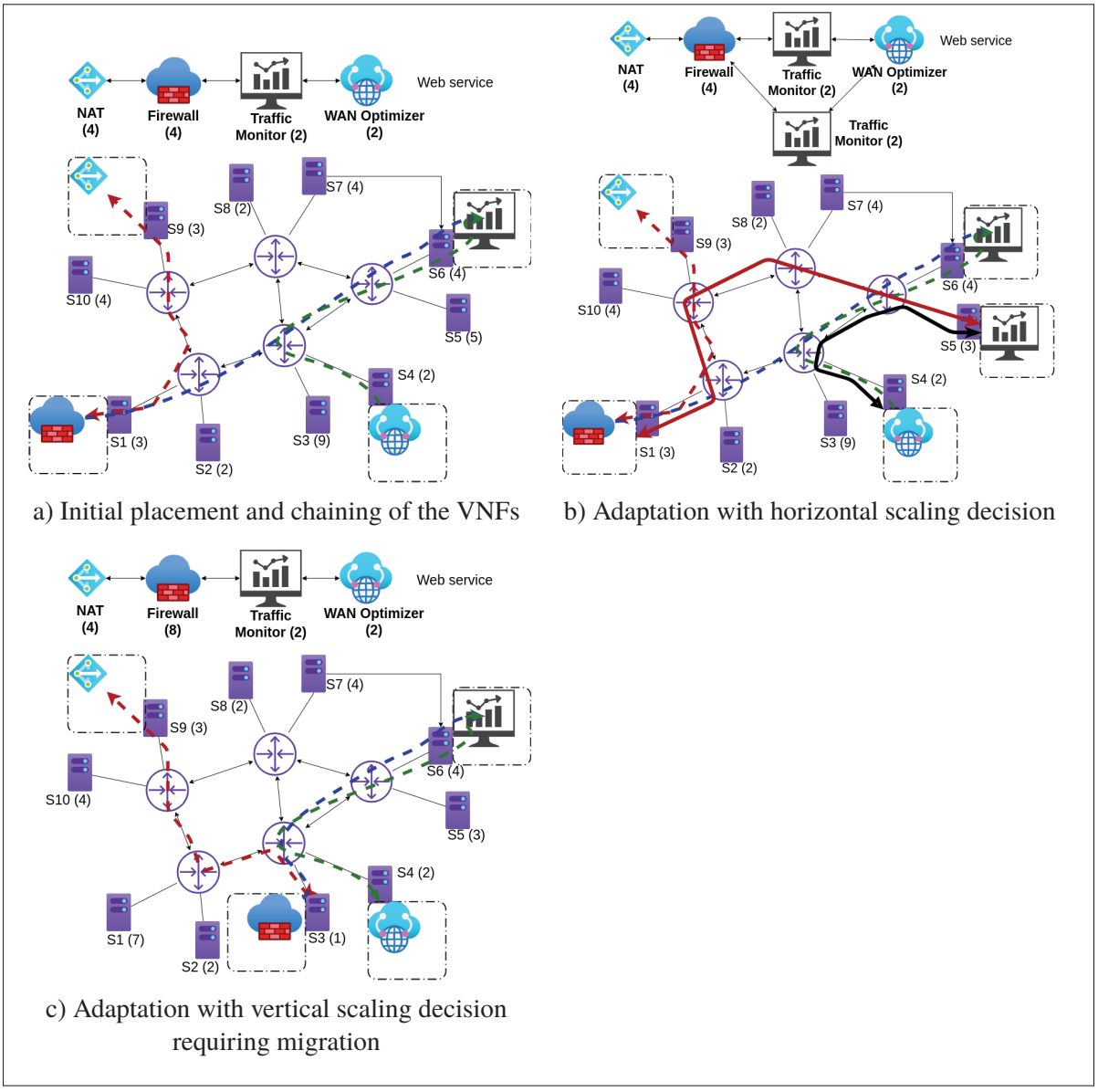
Paper submitted to Future Generation Computer Systems on December 2020

5.1 Abstract

Service function chains, or more generally, virtual network services, evolve over time throughout their life-cycle due to traffic fluctuations, resource usage and the stringent requirements of the SLO (Service Level Objective) that must be fulfilled. Network operators resort to elasticity mechanisms (e.g., vertical and horizontal scaling) and migration procedures to dynamically adapt the service function chains to constantly meet their requirements. However, most state-of-the-art solutions do not evaluate the overall impact of such mechanisms, on the service function chains or the resulting penalty costs (e.g., migration downtime and SLO violation). To bridge this gap, we propose DAVINCI, a decision-making tool with different adaptation policies that allows the network operator to adapt the service function chains with the goal of minimizing network changes (e.g., reallocation of paths and VNFs) while keeping penalty costs at their lowest level. Moreover, DAVINCI allows for the adaptations (e.g., migration, vertical and horizontal scaling) to be expressed as a set of decisions and leverages the Min Cost Flow problem to estimate migration downtime. The analytical evaluation shows that DAVINCI outperforms the existing state-of-the-art solution (NFV-PEAR) in terms of migration and traffic costs, as well as the overhead induced by the migration and elasticity mechanisms, while significantly reducing penalty costs.

Keywords : Network function virtualization, mathematical optimization, orchestration, service function placement and chaining, dynamic adaptation.

5.2 Introduction



a) Initial placement and chaining of the VNFs

b) Adaptation with horizontal scaling decision

c) Adaptation with vertical scaling decision requiring migration

Figure 5.1 Effects of elasticity mechanisms and migration on the placement and chaining of VNFs

Orchestration toolkits are at the heart of cloud systems, and network operators/service providers are taking advantage of the different functionalities and architectures they offer Jungnickel (2013), Sherry, Hasan, Scott, Krishnamurthy, Ratnasamy & Sekar (2012), Weerasiri *et al.* (2017). These orchestration mechanisms are in charge of planning the deployment of virtualized services (i.e. the placement and chaining of the VNFs (Virtual Network Functions)) and provisioning them with the resources needed for their effective operation, as well as adapting the previously established placement and chaining of the VNFs. Moreover, in a dynamic environment, network operators/service providers resort to elasticity mechanisms and migration procedures Al-Dhuraibi, Paraiso, Djarallah & Merle (2018), which are part of the orchestration tools, to dynamically adapt the deployed services and applications to meet their requirements while mitigating the disruptions of individual network functions causing SLO violations. Although these mechanisms offer great flexibility for network operators/service providers, their benefits and overheads must to be carefully studied, not only on a VNF basis, but from the perspective of the whole virtual network service. For consistency, we use SFC to represent the entire Service Function Chain and virtual network service.

Live migration is one of the migration techniques that allow for migrating of virtual machines without turning them off Clark, Fraser, Hand, Hansen, Jul, Limpach, Pratt & Warfield (2005). This technique enables the virtual machines, which execute the virtualized network functions, to continue their executions without service disruptions Cerroni & Callegati (2014). However, it may still cause end-to-end delay and service violations if the new locations of the virtual machines and their paths are not efficiently determined Zhang, Li & Wang (2017), Ibn-Khedher, Abd-Elrahman, Afifi & Forestier (2015). Therefore, migration is not free of charge, and the costs of employing migration (e.g., migration downtime and end-to-end delay violation penalty costs) should be considered Voorsluys, Broberg, Venugopal & Buyya (2009). Authors in Rankothge, Le, Russo & Lobo (2017), Ma, Rankothge, Makaya, Morales, Le & Lobo (2018), Rankothge, Ramalhinho & Lobo (2019) raised several challenges in connection with elasticity mechanisms that require further consideration and study to avoid their unseen costs.

Consider the example of a virtual network service that embodies a web service, for the sake of sketching the concepts highlighted in Figures 5.1a, 5.1b and 5.1c, which illustrate the evolution of a virtual network service topology following the occurrence of horizontal and vertical scaling decisions and migration from its initial state, repsectively. The resource consumption of each VNF and the residual resources of the servers are indicated in brackets, respectively, while solid lines (red and black) and dashed lines (red, green and blue) represent the paths that are either allocated or reallocated to the virtual links connecting the VNFs with respect to the initial placement and chaining of the VNFs, as well as after the occurrence of the adaptation decisions.

From Figure 5.1b, it can be seen that horizontal scaling decision, requires the insertion of a new VNF instance and connecting it to the other VNFs with respect to the original virtual network service topology. In other words, the insertion of the new VNF instance involves finding a feasible assignment that corresponds to a server with sufficient resources and a path connecting it to the other VNFs with sufficient bandwidth capacity that embodies the chaining while load-balancing the traffic among the allocated paths. We assume that an SDN controller acts as a load-balancer to distribute the traffic evenly among the newly set up paths Qazi, Tu, Chiang, Miao, Sekar & Yu (2013). Alternatively, a load-balancing network function may be instantiated to perform the same operation likewise.

As opposed to horizontal scaling, vertical scaling aims to increase resources (e.g., compute resources) at the VNF level or networking resources at the virtual link level. Now, let us discuss a special case of a vertical scaling that may not likely occur within the same server. For practical purposes, when a VNF is in need of additional resources (e.g., CPU cores, memory and storage capacity) and the server that is currently hosting it is unable to afford the demand, the vertical scaling operation is no longer possible. One way to overcome this issue is to allow the vertical scaling operation take place on another server with sufficient residual resources. However, this requires migrating the VNF to another server to enable the vertical scaling operation to take place. Therefore, the paths that originate from the migrated VNF would have to be adjusted. In other words, a new feasible chaining must be established in addition to finding a server to host the migrated VNF.

When migrating a VNF, several changes to the SFC topology occur, as displayed by Figure 5.1c. First, the VNF moves from the actual server to a destination server that must be selected in accordance with the resource requirements of the VNF and the minimization of the associated cost functions (e.g., migration downtime cost). Second, the allocated paths connecting the migrated VNF with its neighbouring VNFs must be refined by searching for alternative link allocations that minimize the end-to-end delay to avoid SLO violations while satisfying the networking constraints in terms of bandwidth capacity. The latter is extremely important, because it must be carried out for each virtual link where the migrated VNF belongs. Similarly, the migrated VNF represents an articulation point Jungnickel (2013), since it disconnects the whole set of link allocations with respect to the SFC topology, which requires all of the paths to be adjusted accordingly.

The dynamic adaptation of an SFC raises several potentially conflicting optimization objectives, let alone elasticity mechanisms and migration procedures. A handful of approaches have studied the migration, vertical and horizontal scaling of VNFs in isolation Cho, Taheri, Zomaya & Bouvry (2017), Xia, Cai & Xu (2016), Houidi, Soualah, Louati, Mechtri, Zeghlache & Kamoun (2017), Wu, Su & Wen (2017), Ibn-Khedher *et al.* (2015), Khai, Baumgartner & Bauschert (2019), Noghani, Kassler & Taheri (2019), Yu, Yang & Fung (2018), Carpio, Jukan & Pries (2018), Zhou, Yi, Wang & Huang (2018), Fangxin Wang, Ruilin Ling, Zhu & Li (2015), Duong-Ba, Nguyen, Bose & Tran (2018), Eramo, Miucci, Ammar & Lavacca (2017b), Eramo, Ammar & Lavacca (2017a), Padhy & Chou (2020), Miotto, Luizelli, Cordeiro & Gaspary (2019), Sugisono, Fukuoka & Yamazaki (2018), Miyazawa, Jibiki, Kafle & Harai (2018), Ghaznavi, Khan, Shahriar, Alsubhi, Ahmed & Boutaba (2015), Duan, Wu, Le, Liu & Peng (2017), Zhang *et al.* (2017), Deric, Varasteh, Basta, Blenk, Pries, Jarschel & Kellerer (2019), Torre, Urbano, Salah, Nguyen & Fitzek (2019), Rankothge *et al.* (2017), Ma *et al.* (2018), Rankothge *et al.* (2019). However, in real-world environments, both processes may take place at the same time to help improve the performance of a virtual network service that is up-and-running. An important limitation of known approaches Cho *et al.* (2017), Xia *et al.* (2016), Houidi *et al.* (2017), Wu *et al.* (2017), Ibn-Khedher *et al.* (2015), Khai *et al.* (2019), Noghani *et al.* (2019), Yu *et al.*

(2018), Carpio *et al.* (2018), Zhou *et al.* (2018), Fangxin Wang *et al.* (2015), Duong-Ba *et al.* (2018), Eramo *et al.* (2017b), Eramo *et al.* (2017a), Padhy & Chou (2020), Miotto *et al.* (2019), Sugisono *et al.* (2018), Miyazawa *et al.* (2018), Ghaznavi *et al.* (2015), Duan *et al.* (2017), Zhang *et al.* (2017), Deric *et al.* (2019), Torre *et al.* (2019), Rankothge *et al.* (2017), Ma *et al.* (2018), Rankothge *et al.* (2019), to the placement and chaining of the VNFs in addition to their reconfigurations (i.e. adaptations), is that they rely on the advantages of these existing mechanisms without analyzing their impacts on the SFC (e.g., VNF and link (re)allocations, end-to-end latency violation, migration downtime).

5.2.1 Key contributions

To fill existing gaps in the research literature, we extended our research work, FASTSCALE Laaziz *et al.* (2019), in which our goal was to enable a fast and scalable placement and chaining of VNFs to dynamically adapt them with the help of elasticity mechanisms and migration procedures. The main contributions of this paper are outlined as follows:

1) We provide DAVINCI as an extended mathematical model to ensure the best adaptation of virtual network services, albeit dynamic changes in resource demands, and/or compute and networking resources as well as traffic changes;
2) DAVINCI integrates the elasticity mechanisms and migration intended for VNFs and realized through the use of decision sets, agnostic to virtualization and infrastructure technologies;
3) DAVINCI enables decision-makers to adopt measures reactively and proactively to anticipate the decisions and their repercussions with least cost;
4) A set of fast and scalable heuristics to cope with the computational complexity of the problem taking advantage of a modified version of Breadth-first Search strategy to speed up the search for a feasible solution;
5) A more detailed experimental evaluation on the effectiveness of DAVINCI against NFV-PEAR in terms of adaptation costs and overheads (e.g., VNF and link (re)allocations).

5.2.2 Structure of the paper

The paper is organized as follows. Section 5.3 presents the related works. Section 5.4 presents the mathematical formulation for the online and dynamic adaptation of virtual network services problem. Our proposed heuristic algorithms are presented in Section 5.5. Section 5.6 discusses the experimental evaluation and results against the state-of-the-art NFV-PEAR approach. Finally, Section 5.7 concludes this paper.

5.3 Related works

In this section, we review the existing substantial approaches that have been proposed thus far in the research literature concerning the adaptation of service function chains, both from analytical and experimental point of view using different methodologies Cho *et al.* (2017), Xia *et al.* (2016), Houidi *et al.* (2017), Wu *et al.* (2017), Ibn-Khedher *et al.* (2015), Khai *et al.* (2019), Noghani *et al.* (2019), Yu *et al.* (2018), Carpio *et al.* (2018), Zhou *et al.* (2018), Fangxin Wang *et al.* (2015), Duong-Ba *et al.* (2018), Eramo *et al.* (2017b), Eramo *et al.* (2017a), Padhy & Chou (2020), Miotto *et al.* (2019), Sugisono *et al.* (2018), Miyazawa *et al.* (2018), Ghaznavi *et al.* (2015), Duan *et al.* (2017), Zhang *et al.* (2017), Deric *et al.* (2019), Torre *et al.* (2019).

Migration toward low network latency reduction in cloud settings was studied in Cho *et al.* (2017), where the authors proposed an ILP formulation and a heuristic to balance the number of times a VNF is migrated with the reduction ratio of network latency that occurs following the migrations. This approach attempts to migrate all of the VNFs to reduce latency. However, the study does not account for the impacts of migrations on the SFC. Furthermore, the proposed approach does not consider complex topologies nor the migration downtime and migration penalty costs.

A flow-based migration technique in the case of a software defined network is proposed in Xia *et al.* (2016). This technique incorporates a migration model cost to be minimized and formulated as an integer linear program. In order to handle the growth in terms of computational complexity, a heuristic algorithm is proposed, for which the migration cost is defined as the

aggregated traffic that is transferred to the controller. However, the proposed approach aims to place all of the migrated VNFs within the same hosts following a consolidation strategy, which is not always feasible and practical. In addition, the previous paths of the flows are not part of the optimization, and therefore, the solution focuses only on the VNFs, ignoring their interconnections. Furthermore, the migration of VNFs is triggered by increasing their allocated resources, which is considered to be a scaling-up operation in essence, but the proposed model fails to consider this.

Authors in Houidi *et al.* (2017) investigated the use of scaling and migration of VNFs for maximizing the provider's revenue by readjusting the host resources of the infrastructure to accept more client's requests and users. First, the problem was formulated as an integer linear program, and then the scaling-up and migration procedures were leveraged by a greedy based heuristic. The aim is to reorganize the location of the VNFs based on their requirements to meet their demands and the traffic load. The proposed solution chooses first to scale-up VNFs before deciding on migrating them as an alternative option. However, one salient and missing aspect relates to the previous placement and chaining of the VNFs that are not part of the optimization when adapting the requests. In fact, the adaptation focused only on choosing between migration and scaling-up VNFs, which maximizes the provider's revenue before the request departs. As a consequence, the overall request is not optimized. In addition, the end-to-end delay and the migration downtime are not considered in the proposed optimization approach.

Authors in Wu *et al.* (2017) proposed TVM as a migration solution to reduce the number of hops between the virtual machines hosting network functions that compose a service function chain, to reduce the latency that is also defined in the SLO. TVM intented to work in cloud data centre settings and works by migrating the virtual machines that cause hop violations onto different physical machines. Therefore, TVM handles hop violations with fewer migrations. However, neither the chaining of the virtual machines nor the cost of migrating the virtual machines is part of the optimization. Yet, the proposed approach works exclusively with peculiar ordered traffic that passes through the virtual machines and limited to linear topologies.

Scalable and efficient algorithms, known as OPAC (Optimal Placement Algorithm for virtual CDN) and HPAC (Heuristic Placement Algorithm for virtual CDN), for the migration of virtual content delivery networks with the help of Gomory-Hu tree transformation algorithm to reduce the search space, were proposed in Ibn-Khedher *et al.* (2015). The aim is to migrate and place the virtual content delivery networks efficiently toward the vicinity of consumers while minimizing the migration, caching and streaming costs and the number of replicas. However, this is a peculiar use case where the aim is to deliver a specific content to be consumed by the end-users which is different from other typical service function chains. In this work, and similarly to ours, the migration of virtual content delivery networks is embodied as a decision that requires finding the optimal target server.

An optimal transition from one mapping to another with the help of migration and the concept of time-expanded networks is proposed in Khai *et al.* (2019). The authors studied how migration would help identify the fewest changes in terms of VNF placement and chaining with the aim of minimizing total migration time. Balancing the reconfiguration cost of service function chains with its energy consumption using migration is proposed in Noghani *et al.* (2019). Different reconfiguration costs were formulated for the migration of stateful VNFs of a given services function chain to determine the best trade-off. The aim of this work is to use migration to improve the quality of the placement and chaining of the VNFs, which may deteriorate over time. Although the work has its own merits, the study lacks generalization, because the authors investigated only simple service function chain topologies (e.g., linear and limited to four VNFs only). Yet, a careful look at the proposed mathematical model suggests that the authors failed to report the type and number of constraints that were considered in their study.

Authors in Rankothge *et al.* (2017), Ma *et al.* (2018), Rankothge *et al.* (2019) proposed using scaling mechanisms to dynamically adapt service function chains to support traffic changes. The proposed approach makes use of a genetic algorithm that aims to minimize the number of used servers and links after scaling the VNFs of the service function chains. However, the proposed approach suffers from excessive execution time for finding a feasible solution, even for a small infrastructure. In addition, the proposed approach only allows for the handling of

linear topologies. Furthermore, the end-to-end delay violation penalty cost is not part of the optimization.

ElasticNFV is proposed as a dynamic resource allocation solution that performs fine-grained vertical scaling over time for service function chains Yu *et al.* (2018). ElasticNFV helps to adjust the allocated resources in real-time while balancing migration time with the embedding cost using first-fit strategy, which reduces the allocated bandwidth for the VNFs. ElasticNFV employs migration when a vertical scaling is unable to occur within certain hosts. However, the proposed framework does not consider the end-to-end delay, and does not minimize the impacts on the whole service function chain VNF placements and path (re)allocations.

Authors in Carpio *et al.* (2018) studied how replication of VNFs can help reduce the number of required migrations. However, the proposed formulation does not consider the migration downtime and penalty costs associated with the violation of the end-to-end delay. In addition, the impacts of migration and replications as well as the current placement and chaining of the VNFs were not part of the optimization.

The effect of live migration of shared VNFs between service function chains is studied by Zhou *et al.* (2018) within the context of software defined networks. The problem is formulated with the aim of minimizing the delay variation between the migrated VNF and its neighbouring VNFs, taking into account the actual placement of VNFs and the target location of the VNFs. However, the underlying link (re)allocations are not part of the optimization when migrating the VNFs, since the migrated VNFs are assumed to be shared between service function chains.

Authors in Fangxin Wang *et al.* (2015) proposed a preplanned allocation approach with bandwidth guarantee to solve the problem of VNF placement and scaling, which allows tenants to specify different periods for which it is not necessary to execute the elastic allocation method in order to handle various traffic workloads. They formulated the problem as an integer linear program, and proposed a heuristic that exploits the underlying data centre network topology to minimize the bandwidth allocation. However, the proposed approach is tied to the data centre network topology, where the goal is to save bandwidth allocation for the placed VNFs while minimizing

the number of migrations. Furthermore, the migration downtime and the end-to-end delay violation penalty costs are not part of the optimization.

Authors in Duong-Ba *et al.* (2018) used migration to consolidate virtual machines in a data centre to accept more virtual machine requests placement. The problem is formulated as an optimization problem and a relaxed form is leveraged to form a near-optimal solution, where the aim is to minimize the energy spent on migration. However, the authors considered a scenario where the cloud provider has enough capacity to serve all of the requests, including the new virtual machines to be placed, which is not realistic given the scarcity of resources.

Authors in Eramo *et al.* (2017b) investigated solutions for reducing energy consumption and rejected service function chain requests due to the lack of bandwidth based on vertical scaling and migration. The problem is formulated as an integer linear program, and heuristics were proposed to tackle the computational complexity of the optimal solution. The proposed approaches aim to find the required number of CPU cores for the placement and chaining of the VNFs during peak hours, along with the migration policy that minimizes the energy and reconfiguration costs. However, the proposed approaches try to balance energy consumption with revenue loss due to the quality of service degratation arising with mirgation with the help of Markov decision process. The same approach is extended to account for low traffic periods and decide when and where to migrate the VNFs in order to minimize energy consumption accounting for migration and consolidation Eramo *et al.* (2017a). Nevertheless, the authors considered a scenario driven by the case where consolidation is the main building-block of the proposed solutions to minimize the energy consumption while being limited to linear topologies. Yet, the end-to-end delay violation penalty cost is not part of the optimization.

Authors in Padhy & Chou (2020), studied the problem of traffic changes in service function chaining, and proposed an approach that helps the orchestrator decide among the elasticity mechanisms and migration procedures to choose from with lowest cost in terms of overheads and energy. According to the study, the scale-out is the most preferred approach when the traffic is

increasing significantly. However, the authors did not consider the SLO violation and migration downtime penalty costs or the chaining of the VNFs within the optimization model.

NFV-PEAR Miotto *et al.* (2019) is proposed as a framework that allows network operators to adaptively (re)arrange the placement and chaining of VNFs in response to migration and scaling decisions. NFV-PEAR is intended to maintain the end-to-end delay within the specified maximum limit, despite traffic and resource fluctuations. The proposed formulation attempts to reduce the network changes (e.g., reallocation of VNFs and network flows) while instantiating fewer VNFs to process the increasing traffic and load on the service function chain. However, the proposed formulation does not consider the migration downtime and penalty costs associated with the violation of the end-to-end delay, because it is considered to be a constraint. In addition, the impacts of migration over the service function chain are not studied compared to our work.

A method for simultaneous VNF migrations is proposed in Sugisono *et al.* (2018) for reducing flow termination and flow-quality degradations due to live migration. A technique that uses resource arbitration with migration to avoid CPU saturation for VNFs is proposed in Miyazawa *et al.* (2018). Authors in Ghaznavi *et al.* (2015) used migration and scaling mechanisms to handle traffic fluctuations while considering host and bandwidth resources. In contrary with Ghaznavi *et al.* (2015), authors in Duan *et al.* (2017) used scale-out mechanism for VNFs in an IMS-based system, and proposed a technique that deploys them in data centres that possess high residual resources.

From a system perspective, a technique with low overheads (e.g., migration downtime, total migration time) is proposed in Zhang *et al.* (2017), which exploits intel's fast-compression and para-virtualization technologies to perform VNF migrations. A demo is presented in Deric *et al.* (2019), which showcases the benefits of using migration to move a firewall close to the end-user, and how it is supported in a virtualized software defined network using an existing hypervisor. Authors in Torre *et al.* (2019), with a realistic experiment using Docker containers, demonstrated the importance of considering the host and networking resources when performing

live migration of the VNFs. If not considered, the overall performance of the migration will deteriorate, leading to higher service downtime and migration time.

Compared to the aforementioned research works, in this paper, we address, through DAVINCI, the trade-offs between performance (e.g., VNF and link (re)allocations) and penalty costs (e.g., SLO and migration downtime violations) that raise when applying elasticity mechanisms and migration techniques to dynamically adapt virtual network services. We show that different adaptation policies may result in different costs.

5.4 Problem Formulation

We begin by formally defining the DAVINCI problem, and then we elaborate a Mixed Integer Linear Program (MILP) formulation for it and prove its NP-Hardness. Unless otherwise specified. Table 5.1 summarizes the list of symbols used throughout this paper.

5.4.1 DAVINCI Problem

In the DAVINCI problem, the residual resource capacities of compute and networking resources, the previous placement and chaining of the VNFs and the adaptation decisions for each VNF are provided as inputs. A stream of virtual network services is also provided for which the MILP formulation will decide the optimal new placement and chaining of the VNFs.

The usage of resources varies over time, and the adaptation decisions differ for each virtual network service. An answer to the DAVINCI problem must take advantage of these facts to diminish the effects of the changes (e.g., link allocations, migration of VNFs) that may occur on the SFC generating SLO violation penalty costs. We assume that horizontal and vertical scaling and migration decisions are made at particular time periods. These time periods are identified by the occurrence of one or more of the following possible event triggers, including changes in the amount of occupied resources by a VNF on a server, traffic fluctuations causing higher processing delays or a server running a couple of VNFs that is becoming overloaded, to name a few. The MILP makes online and dynamic adaptations based on these events. The objective of

the MILP is to minimize the total cost due to adaptation of the VNFs comprising migration, vertical and horizontal scaling costs, the total traffic cost due to changes in the link allocations based on the residual compute and networking resources, while minimizing the SLO violation and migration downtime penalty costs. Finally, the proposed MILP leverages the Min-Cost flow problem to provide an estimate for the migration downtime as the delay from the origin node to the target node where the VNF is reallocated. This estimation is considered to be the lower limit. Also, it is assumed to be the perceived migration downtime by the neighbouring VNFs (i.e., interconnected with the migrated VNF) according to the topology of the virtual network service.

5.4.1.1 Theorem

The DAVINCI problem is NP-Hard.

5.4.1.2 Proof

Proof. At each occurrence of a decision adaptation, we must solve the general service function chain orchestration problem Gil Herrera & Botero (2016), Bonfim *et al.* (2019a), which is known to be NP-Hard. The general service function chain orchestration problem can be reduced to the DAVINCI problem by assuming that the service function chain request is a new request that must be deployed with location constraints Luizelli *et al.* (2015a). In this setting, a solution to the DAVINCI problem will also be a solution to the general service function chain orchestration problem, and therefore, the DAVINCI problem is NP-Hard. In fact, the DAVINCI problem is computationally much harder than the general service function chain orchestration problem, because migration downtime and SLO violation penalty costs must be part of the optimization while adapting the request.

5.4.2 Physical Infrastructure

The physical infrastructure is modeled as a graph denoted by $G = (V, E)$ where V represent the set of server nodes and E represents the set of interconnections between the server nodes. Each

Table 5.1 Glossary of Symbols

$G_{vns} = (V_{vns}, E_{vns})$	Denotes the virtual network service request.
$G = (V, E)$	Denotes the physical network.
$Bw^{(u,v)}$	Denotes the amount of bandwidth required between VNFs u and v.
$\Delta_{(x,y)}$ and $C^{(x,y)}$	Denote the delay and link capacity between nodes x and y, respectively.
Req_u^X and C_x^Y	Denote the required resource X (e.g., CPU, Memory and Storage) of VNF u and residual resource Y (e.g., CPU, Memory and Storage) of node x, respectively.
γ	Denotes the SLO violation cost (e.g. \$), per unit of time, due to end-to-end delay.
$\lambda_{(u,v)}$	Denotes the Lagranian multiplier with respect to virtual link (u, v) connecting VNFs u and v.
$D_{(x,y)}$	Denotes the number of hops between nodes x and y.
$\theta^{(u,v),max}$	Denotes the maximum end-to-end delay tolerated between VNFs u and v.
δ	Denotes the penalty cost (e.g., \$), per unit of time, associated with the violation of the maximum tolerated downtime due to migration.
N_u	Denotes the neighbouring VNFs of VNF u that are experiencing migration downtime.
Λ	Denotes the monetary cost to transport traffic of the hosted VNFs through one physical hop distance.
ρ	Denotes the cost of adding one unit of CPU, one unit of memory and one unit of storage capacity with respect to vertical scaling.
Γ_x	Denotes a monetary cost to instantiate/operate one VNF on a server x.
ϵ_u^s	Denotes a penalty cost of migrating a VNF u with respect to a given strategy s.
ω_u^t	Denotes a penalty cost of migrating a VNF u given its type t.
$\nabla_{dst(u)}^{src(u)}$	Denotes the total delay computed from the current server where the VNF is located i.e. $src(u)$ to the destination server where the VNF is migrated i.e. $dst(u)$.
$I_{Mig}(u)$	Denotes the migration decision set for VNF u.
$I_{VS}(u)$ and $I_{HS}(u)$	Denote vertical and horizontal scaling indicator functions for VNF u.
M, VS and HS	Denote migration, vertical and horizontal scaling decision sets.
$\widehat{\alpha_{u,x}^{vns}}$, $\alpha_{u,x}^{vns} \in \{0, 1\}$	$\widehat{\alpha_{u,x}^{vns}} = 1$ if VNF u of a virtual network service vns is up-and-running on server x and 0 otherwise. Whereas, $\alpha_{u,x}^{vns} = 1$ if server x is candidate to host VNF u of a virtual network service vns and 0 otherwise.
$\widehat{\beta_{(x,y)}^{(u,v)}}, \beta_{(x,y)}^{(u,v)} \in \{0, 1\}$	$\widehat{\beta_{(x,y)}^{(u,v)}} = 1$ if the traffic of VNFs u and v is using physical link (x, y) and 0 otherwise. Whereas, $\beta_{(x,y)}^{(u,v)} = 1$ if physical link (x, y) can carry the traffic of VNFs u and v and 0 otherwise.

interconnection (x, y) is associated with a delay $\delta_{(x,y)}$, a number of hops $D_{x,y}$ and a residual bandwidth capacity $C^{(x,y)}$. Each server node is associated with a vector of residual resources (e.g., CPU, Memory, Storage, Power consumption).

5.4.3 Virtual network service

The virtual network service is represented as a graph of any type of topology and any size, and denoted as $G_{vns} = (V_{vns}, E_{vns})$. Each VNF asks for a vector of resources (e.g., CPU, Memory, Storage) and each interconnection between a pair of VNFs asks for an amount of bandwidth $Bw^{(u,v)}$ and end-to-end delay $\theta^{(u,v),max}$ that must be fulfilled over its lifespan to deliver the desired performance with the quality of service needed to avoid incurring a higher SLO violation penalty cost. Without loss of generality, a VNF may belong to various types (e.g., Firewall, IPS, Proxy) and in the case of IMS (e.g., P-CSCF, I-CSCF and S-CSCF), to name a few.

Let $I_{Mig}(u)$, $I_{VS}(u)$ and $I_{HS}(u)$ be the binary indicator functions that designate, the migration, vertical and horizontal scaling decisions for each VNF u with respect to each virtual network service. In the light of that, we set up the decision sets M, VS and HS, respectively. Mention should be made of the *sine qua non* of $M \cap VS \cap HS = \emptyset$. In practical terms, a VNF u is unable to be in need of migration, vertical and horizontal scaling concurrently. DAVINCI needs to know the decisions resulting from the indicator functions to determine the best reconfiguration under the frame of placement and chaining of the VNFs. This makes it possible for DAVINCI to be initiated reactively and proactively (e.g., coping with traffic fluctuations) and arrange the virtual network service to assert its execution under the same operating conditions.

5.4.4 Decision variables

Previous placement of VNFs: $\widehat{\alpha_{u,x}^{vns}} = 1$ if VNF u of a virtual network service vns is up-and-running on server x and 0 otherwise.

Previous Chaining of VNFs: $\widehat{\beta_{(x,y)}^{(u,v)}} = 1$ if the traffic of VNFs u and v is using physical link (x, y) and 0 otherwise.

New Placement of VNFs: $\alpha_{u,x}^{vns}$ = 1 if server x is candidate to host VNF u of a virtual network service vns and 0 otherwise.

New Chaining of VNFs: $\beta_{(x,y)}^{(u,v)}$ = 1 if physical link (x, y) can carry the traffic of VNFs u and v and 0 otherwise.

Previous and new placement and chaining variables, respectively, capture the previous configuration, denoted by $\Xi(\widehat{\alpha_{u,x}^{vns}}, \widehat{\beta_{(x,y)}^{(u,v)}})$ and the new configuration, outlined by $\Xi(\alpha_{u,x}^{vns}, \beta_{(x,y)}^{(u,v)})$, owing to the decision adaptations (e.g., migration, horizontal and vertical scaling) in reference to the virtual network services.

5.4.5 Objective functions

An optimal adaptation should consider several optimization metrics, including migration, horizontal and vertical scaling costs, the traffic cost, the total migration donwtime and SLO violation costs. Therefore, a solution must balance the aforementioned metrics as follows:

1) $\mathcal{F}_1$ minimizes the total cost due to the adaptation of VNFs of the whole SFC, which comprises the migration, vertical and horizontal scaling costs;
2) $\mathcal{F}_2$ minimizes the total traffic cost due to the adaptation of VNFs of the entire virtual network service;
3) $\mathcal{F}_3$ minimizes the total migration downtime cost experienced by the migration of VNFs with respect to the migration decision set M;
4) $\mathcal{F}_4$ minimizes the total SLO violation cost experienced by the adaptation (e.g., migration, vertical and horizontal scaling) of the VNFs of the whole virtual network service. With the help of Lagrange Multipliers (LM) that transform constraint 5.9 into a cost function Fisher (1981a), we are able to quantify the SLO violation penalty cost;

   Formally, the aforementioned cost functions ($\mathcal{F}_1$, $\mathcal{F}_2$, $\mathcal{F}_3$ and $\mathcal{F}_4$) are defined as follows:

$$\mathcal{F}_1 = Mig\_cost + H\_Scaling\_cost + V\_Scaling\_cost \tag{5.1}$$

where :

$$Mig\_cost = \sum_{\forall u \in M, \forall x} \alpha_{u,x}^{vns} \times (1 - \widehat{\alpha_{u,x}^{vns}}) \times I_{Mig}(u) \times (\omega_u^t + \epsilon_u^s) + \sum_{\forall u \in VS, \forall x} \alpha_{u,x}^{vns} \times (1 - \widehat{\alpha_{u,x}^{vns}}) \times I_{VS}(u) \times (\omega_u^t + \epsilon_u^s)$$

$$H\_Scaling\_cost = \sum_{\forall u \in HS, \forall x} \alpha_{u,x}^{vns} \times \Gamma_x \times I_{HS}(u)$$

$$V\_Scaling\_cost = \sum_{\forall u \in VS, \forall x} \alpha_{u,x}^{vns} \times (1 - \widehat{\alpha_{u,x}^{vns}}) \times I_{VS}(u) \times \rho + \sum_{\forall u \in VS, \forall x} \alpha_{u,x}^{vns} \times (1 - \widehat{\alpha_{u,x}^{vns}}) \times I_{VS}(u) \times \rho$$

$$\mathcal{F}_2 = \sum_{\forall (x,y) \in E} D_{x,y} \sum_{\forall (u,v) \in E_{vns}} Bw_{(u,v)}^{vns} \times \beta_{(x,y)}^{(u,v)} \tag{5.2}$$

$$\mathcal{F}_3 = \sum_{\forall u \in M, \forall x} \alpha_{u,x}^{vns} \times (1 - \widehat{\alpha_{u,x}^{vns}}) \times I_{Mig}(u) \times \nabla_{dst(u)}^{src(u)} \times N_u \times \delta + \sum_{\forall u \in VS, \forall x} \alpha_{u,x}^{vns} \times (1 - \widehat{\alpha_{u,x}^{vns}}) \times I_{VS}(u) \times \nabla_{dst(u)}^{src(u)} \times N_u \times \delta \tag{5.3}$$

$$\mathcal{F}_4 = \sum_{\forall (u,v) \in E_{vns}} \lambda_{u,x} \times [\theta^{(u,v),max} - \sum_{\forall (x,y) \in E} \beta_{(x,y)}^{(u,v)} \times \delta_{(x,y)}] \times \gamma \tag{5.4}$$

5.4.6 Constraints

Constraints 5.5, 5.6 and 5.7 ensure that physical residual resources of the servers are not overcommitted after adaptation. Constraint 5.8 refers to the limitation of each physical link that should not be exceeded by the traffic of each pair of VNFs after adaptation. Constraint 5.9 checks that each delay of a virtual link is met to avoid violating the SLO. Constraint 5.10 builds the path interconnecting the VNFs with respect to their placement and how they are linked according to G_{vns} after adaptation. Constraint 5.11 specifies that a VNF instance is placed on only one server node. Constraint 5.12 states that the location of all of the VNFs, other than those requiring adaptation with respect to the decision sets, should be kept unchanged. Constraint

5.13 provides the VNFs, with a new server location with respect to the migration decision set. Constraint 5.14 is for the positivity constraint on the decision variables.

5.4.7 Adaptation policies

From now on, we define DAVINCI_LM (Lagrange Multipliers) by $\mathcal{F}_1$, $\mathcal{F}_2$, $\mathcal{F}_3$ and $\mathcal{F}_4$ subject to constraints 5.5, 5.6, 5.7, 5.8, 5.10, 5.11, 5.12, 5.13 and 5.14. In addition to DAVINCI_-LM, we define the following variants DAVINCI, DAVINCI_FLEX_1, DAVINCI_FLEX_2 and DAVINCI_FLEX_3 omitting $\mathcal{F}_4$.

Unless otherwise specified, we describe DAVINCI (resp. DAVINCI_LM), DAVINCI_FLEX_1 (resp. DAVINCI_LM_FLEX_1), DAVINCI_FLEX_2 (resp. DAVINCI_LM_FLEX_2) and DAVINCI_FLEX_3 as the adaptation policies.

1) DAVINCI with $\mathcal{F}_1$, $\mathcal{F}_2$ and $\mathcal{F}_3$ subject to constraints 5.5, 5.6, 5.7, 5.8, 5.10, 5.11, 5.12, 5.13 and 5.14;
2) DAVINCI_FLEX_1 with $\mathcal{F}_1$, $\mathcal{F}_2$ and $\mathcal{F}_3$ subject to constraints 5.5, 5.6, 5.7, 5.8, 5.10, 5.11, 5.12 and 5.14. However, constraint 5.13 becomes:

$$\forall u \in V_{vns} \setminus \{M, HS\}, \forall v \in V : \alpha_{u,x}^{vns} = \widehat{\alpha_{u,x}^{vns}}$$

3) DAVINCI_FLEX_2 with $\mathcal{F}_1$, $\mathcal{F}_2$ and $\mathcal{F}_3$ subject to constraints 5.5, 5.6, 5.7, 5.8, 5.10, 5.11, 5.12 and 5.14;
4) DAVINCI_FLEX_3 with $\mathcal{F}_1$, $\mathcal{F}_2$ and $\mathcal{F}_3$ subject to constraints 5.5, 5.6, 5.7, 5.8, 5.10, 5.11, 5.12 and 5.14. However, constraint 5.13 becomes:

$$\forall u \in V_{vns} \setminus \{M\}, \forall v \in V : \alpha_{u,x}^{vns} = \widehat{\alpha_{u,x}^{vns}}$$

In contrast to 1)), 2)) and 3)), we define DAVINCI_LM_FLEX_1, DAVINCI_LM_FLEX_1, DAVINCI_LM_FLEX_2, respectively, plus $\mathcal{F}_4$.

$$\forall v \in V : \sum_{u} \alpha_{u,x}^{vns} \times Req_{u}^{CPU} \leq C_{x}^{CPU} \tag{5.5}$$

$$\forall x \in V : \sum_{u} \alpha_{u,x}^{vns} \times Req_{u}^{Memory} \leq C_{x}^{Memory} \tag{5.6}$$

$$\forall x \in V : \sum_{u} \alpha_{u,x}^{vns} \times Req_{u}^{Storage} \leq C_{x}^{Storage} \tag{5.7}$$

$$\forall (x,y) \in E : \sum_{(u,v)} \beta_{(x,y)}^{(u,v)} \times Bw^{(u,v)} \leq C^{(x,y)} \tag{5.8}$$

$$\forall (u,v) \in E_{vns} : \sum_{(x,y)} \beta_{(x,y)}^{(u,v)} \times \Delta_{(x,y)} \leq \theta^{(u,v),max} \tag{5.9}$$

$$\forall (x,y) \in E, \forall x \in V : \\ \sum_{x} \beta_{(x,y)}^{(u,v)} - \sum_{x} \beta_{(y,x)}^{(u,v)} = \alpha_{u,x}^{vns} - \alpha_{v,x}^{vns} \tag{5.10}$$

$$\forall u \in V_{vns} : \sum_{x} \alpha_{u,x}^{vns} = 1 \tag{5.11}$$

$$\forall u \in V_{vns} \setminus \{VS, M, HS\}, \forall v \in V : \alpha_{u,x}^{vns} = \widehat{\alpha_{u,x}^{vns}} \tag{5.12}$$

$$\forall u \in M, \forall x \in V : \alpha_{u,x}^{vns} + \widehat{\alpha_{u,x}^{vns}} \leq 1 \tag{5.13}$$

$$\forall u \in V_{vns}, \forall x \in V : \alpha_{u,x}^{vns}, \widehat{\alpha_{u,x}^{vns}}, \beta_{(x,y)}^{(u,v)}, \widehat{\beta_{(x,y)}^{(u,v)}} \in \{0,1\} \tag{5.14}$$

5.4.8 Min-Cost flow problem for migration downtime

Algorithm 5.1 describes the computation method that we rely on to estimate the migration downtime, which we consider to be the time it takes to migrate the VNF from the source to the destination as a lower limit. The algorithm takes as inputs the source location of the VNF ($src(u)$) that must be migrated and its future target location ($dst(u)$) as inputs, along with the current state of the infrastructure in terms of delays between each pair of servers ($\Delta_{(x,y)}$). Then, we feed an instance of the Min-Cost-flow problem Medhi (2004) with the abovementioned inputs to perceive the migration downtime estimate from the source location to the destination location of the migrated VNF u i.e. $\nabla_{dst(u)}^{src(u)}$.

Algorithm 5.1 Migration downtime estimation

Input: $src(u)$, $dst(u)$ and $\Delta_{(x,y)}$
Output: $\nabla_{dst(u)}^{src(u)}$
1: $\nabla_{dst(u)}^{src(u)}$ = Solve Min-Cost flow problem ($src(u)$, $dst(u)$, $\Delta_{(x,y)}$)
2: **return** $\nabla_{dst(u)}^{src(u)}$

5.5 Heuristic algorithm

To cope with the computational complexity of the proposed MILP formulation, we develop a fast heuristic to solve medium to large instances of our problem. In this section, we first give an overview of the modified Breadth-first search (BFS) strategy that we leverage to speed up the search for a feasible solution as it runs in $O(V + E)$ where V and E represent the set of vertices and edges, respectively. The rationale behind the use of BFS is that it enables us chain the VNFs quickly compared to Dijkstra, which runs in $O(E \log V)$ using priority queues Cormen (2009). We then describe the main steps of our proposed heuristics.

Algorithm 5.2 features the well-known Breadth-first search algorithm, which allows us to pass over a tree or a *graph* G=(V, E) given as input in addition to a root node Cormen (2009). The original algorithm starts at the *root* node and explores all of the neighbouring nodes at the current depth level before moving on to the nodes at the next depth level (line 14-33). Basically, we added the *target* node and the *adjacency list* as inputs to help build a path (lines 16-32) between the *root* node (a *source* node) and the *target* (a *destination* node) (line 23). We refer the reader to the original algorithm Cormen (2009) to grasp its functioning before understanding the modification that we inserted. Moving to the core of the heuristic, Algorithm 5.3 showcases the different steps toward an adaptation of a given virtual network service. It takes the request, the set of nodes and links, the previous location of the VNFs and their chaining, the set of decisions and the adjacency list as inputs in order to provide us with the new placement and chaining of the VNFs. Basically, we loop through the set of VNFs (lines 4-33), and for each VNF within the set of decisions (lines 5-25), we perform the required measures (e.g., migration, vertical and

Algorithm 5.2 modified Breadth-first search (mBFS)

```
Input: root, target, G=(V, E) and adjacenyList
Output: a path connecting root with target
 1: distance ← {}
 2: predecessor ← {}
 3: marked ← {}
 4: path ← []
 5: end ← False
 6: while vertex in V do
 7:     marked[vertex] ← False
 8:     predecessor[vertex] ← 0
 9: end while
10: distance[root] ← 0
11: predecessor[root] ← None
12: vertexqueue ← Queue
13: vertexqueue.put(root)
14: while vertexqueue not ∅ and not end do
15:     currentVertex ← vertexqueue.get()
16:     for v in adjacenyList[currentVertex] do
17:         if not marked[v] then
18:             if v != root then
19:                 predecessor[v] ← currentVertex
20:                 distance[v] ← distance[currentVertex]+1
21:             end if
22:             if v == target then
23:                 path ← path ⋃ (currentVertex, v)
24:                 end ← True
25:                 break
26:             end if
27:             vertexqueue.put(v)
28:             marked[v] ← True
29:         end if
30:     end for
31: end while
32: return path
```

horizontal scaling) (lines 6, 14 and 22). If the VNF requires migration (line 6), vertical scaling (line 14) or horizontal scaling (line 22), then we execute Algorithm 5.4 to find a new server location. If Algorithm 5.4 succeeds, we update the location of the VNF with the newly obtained

Algorithm 5.3 Fast and scalable adaptation (fast_DAVINCI_flex_3)

```
Input: n_request, nodes, links, previous_locations, previous_chaining, decisions and adjacenyList
Output: new_vnfs_with_locations and new_chaining
 1: new_vnfs_with_locations ← {}
 2: new_chaining ← {}
 3: rejected ← False
 4: for vnf in previous_locations and not rejected do
 5:     if vnf in decisions then
 6:         if decisions[vnf] == 0 then
 7:             if Output(Algorithm 5.4) not None then
 8:                 new_vnfs_with_locations ← candidateNode
 9:             else
10:                 rejected ← True
11:                 break
12:             end if
13:         end if
14:         if decisions[vnf] == 1 and not rejected then
15:             if Output(Algorithm 5.4) not None then
16:                 new_vnfs_with_locations ← candidateNode
17:             else
18:                 rejected ← True
19:                 break
20:             end if
21:         end if
22:         if decisions[vnf] == 2 and not rejected then
23:             if Output(Algorithm 5.4) not None then
24:                 new_vnfs_with_locations ← candidateNode
25:             else
26:                 rejected ← True
27:                 break
28:             end if
29:         else
30:             new_vnfs_with_locations[vnf] ← previous_locations[vnf]
31:         end if
32:     end if
33: end for
34: if not rejected then
35:     for vlink in n_request[1] do
36:         scr_vnf ← new_vnfs_with_locations[vlink[0]]
37:         dst_vnf ← new_vnfs_with_locations[vlink[1]]
38:         if src_vnf = dst_vnf then
39:             new_chaining ← 0
40:         end if
41:         if (src_vnf, dst_vnf) in links then
42:             new_chaining[vlink] ← (src_vnf, dst_vnf)
43:         end if
44:         if (src_vnf, dst_vnf) not in links then
45:             new_chaining[vlink] ← Output(Algorithm 5.2)
46:         end if
47:     end for
48:     new_chaining ← new_chaining ⋃ previous_chaining
49: end if
50: return new_vnfs_with_locations, new_chaining
```

server location (Algorithm 5.4, lines 8, 16 and 24) for the migration, vertical and horizontal scaling decisions, respectively. Otherwise, the request is rejected (lines 10, 18 and 26). For the VNFs that do not require adaptation, their server locations are kept unchanged (line 30). Lines

Algorithm 5.4 Candidate Search

```
Input: n_request, nodes, vnf
Output: candidateNode
 1: found ← False
 2: for candidateNode in nodes do
 3:     if n_request[vnf].CPU ≤ candidateNode.CPU and n_request[vnf].RAM ≤
        candidateNode.RAM and n_request[vnf].STORAGE ≤ candidateNode.STORAGE
        and not found then
 4:         new_vnfs_with_locations ← candidateNode
 5:         found ← True
 6:         break
 7:     end if
 8: end for
 9: if found then
10:     return CandidateNode
11: else
12:     return None
13: end if
```

38 and 46 are the special cases where the VNFs are either located within the same server or the servers are adjacent to each other. The chaining of the new VNFs is performed with the help of Algorithm 5.2, while the VNFs that do not require adaptations their chaining are kept unchanged (line 48). Finally, we merge the previous chaining of the VNFs with that of the new relocated VNFs to form the new chaining (line 48). Similarly, we devise fast_DAVINCI as another variant of DAVINCI, which is the same as fast_DAVINCI_flex_3, except that for the VNFs that require horizontal scaling, their locations are kept unchanged. Consequently, lines 22-29 are absent in this variant.

5.5.1 Theorem

The proposed heuristic i.e. Algorithm 5.3 runs in $\approxeq O(V + E)$.

5.5.2 Proof

Proof. The heuristic is invoked when a VNF or a group of VNFs require adaptations (lines 4-5). Therefore, lines 4-33 are executed $O(V_{vns} \times (V))$, because we must call Algorithm 5.4 depending on the type of adaptation (e.g., migration, vertical and horizontal scaling). Moving to the chaining, lines 34-49 runs in $O(E_{vns} \times (V + E))$, because we must call Algorithm 5.2 each time we must to (re)chain the adapted VNFs. Thus, the heuristic runs in $O(V_{vns} \times V + E_{vns} \times (V + E))$. Finally, the overall heuristic runs in $\approxeq O(V + E)$.

5.6 Evaluation

We compare and contrast the performances of our proposed approaches against NFV-PEAR Miotto *et al.* (2019), which is a better competitive candidate, using different scenarios, request types and adaptation policies. However, mention should be made of the source code of the other reviewed approaches, which were not made available at the time of writing this paper. Also, most of the papers reviewed in Section 5.3 have insufficient detailed information which would have been mandatory to back-develop the proposed solutions from square one. Nevertheless, we had recourse to code NFV-PEAR from a to z following the concise description provided by the authors in Luizelli *et al.* (2015a), Miotto *et al.* (2019), Miotto (2017).

In the rest of this section, we first present the datasets in Section 5.6.1, then the performance metrics in Section 5.6.2, followed by the simulation setup in Section 5.6.3. Finally, a discussion of the experimental results is presented in Section 5.6.4.

5.6.1 Datasets

5.6.1.1 Virtual network service requests

We synthetically generated, 30 requests per scenario for a fair comparison with Miotto *et al.* (2019). Each request comprises 5 VNFs with different requirements in terms of resources,

where the number of CPU cores, Memory (in Gb) and storage capacity (in Gb) were uniformly drawn in the ranges [1, 6], [1, 12] and [8, 128], respectively. Furthermore, the set of decisions for each type of evaluated request, comprising migration, vertical and horizontal scaling, and requirements in terms of end-to-end delay and bandwidth, are summarized in Tables 5.2 and 5.3, respectively. Three main scenarios have been defined for non-sensitive, sensitive and highly sensitive SFCs in terms of end-to-end latency. For instance, highly sensitive SFCs require an end-to-end latency between 1 and 5 ms. Other values are considered, because they are inputs for DAVINCI (e.g., Table 5.3).

Table 5.2 Decision adaptations

Scenarios	Decision adaptations		
	Migration	Vertical scaling	Horizontal scaling
Non sensitive	16	12	12
Sensitive	8	9	11
Highly sensitive	16	12	7

Table 5.3 Requests requirements per scenario

Requirements of the requests	Scenarios		
	Non sensitive	Sensitive	Highly sensitive
End-to-end delay	[20, 50] ms	[5, 10] ms	[1, 5] ms
Bandwidth	[200, 500] Mbps		

5.6.1.2 Network Topologies

We have considered two types of networks: a small network and Fat-Tree Data Centre network topologies. Table 5.4 summarizes the characteristics of the different topologies in terms of available resources and servers count.

5.6.2 Performance Metrics

In this paper, seven metrics are evaluated.

Table 5.4 Network characteristics

Scenarios \ Network characteristics	Small network	Fat Tree
Server count	20	250
CPU	70	70
Memory	1TB	1TB
Storage capacity	1TB	1TB
Link delay	[2, 6] ms	[10, 50] us
Link capacity	10 Gbps	10 Gbps

1) **Execution time**: We measure the time taken by a given approach to find a solution for the dynamic adaptation of the requests;
2) **Traffic cost**: We calculate the cost of the total amount of traffic allocated due the dynamic adaptation of the requests, as defined by $\mathcal{F}_2$;
3) **SLO violation and migration downtime costs**: We compute the SLO violation and migration downtime costs due to the adaptation of the requests as defined by $\mathcal{F}_3$ and $\mathcal{F}_4$, respectively;
4) **Migration, horizontal and vertical scaling costs**: We compute the migration, vertical and scaling costs that occur due the adaptation of VNFs, as defined by $\mathcal{F}_1$;
5) **Reduced and allocated links**: We examine the total number of allocated links of all of the adapted requests in the new configuration and the total number of reduced links of all of the adapted requests from the previous configuration. Recall that a configuration comprises the placement of VNFs and their chaining;
6) **Extra migrated VNFs**: We assess the number of additional migrated VNFs that are not part of the migration decision set with respect to each approach and strategy;
7) **Accepted and rejected adaptations**: We report the number of successfully adapted and rejected requests with respect to the decision adaptations;

5.6.3 Simulation Setup

A set of experiments were conducted to evaluate the different approaches as well as the elaborated policies in terms of the performance metrics described in Section 5.6.2. In all experiments, we generated per request type (i.e. non sensitive, sensitive and highly sensitive) a set of 30 requests and then deployed them to reflect a set of up and running SFCs. The same procedure is applied for each network topology.

In all of the experiments, we first deployed 30 requests for each request type (e.g., non-sensitive, sensitive and highly sensitive) in order to simulate a set of virtual services that are already deployed and up-and-running, and then generated the set of decisions to adapt them. Each decision set was generated arbitrarily, and each set contains decisions for each VNF (e.g., migration, vertical and horizontal scalings). For coherence, a virtual network function admits one adaptation at a time (e.g., migration, vertical or horizontal scaling decision). NFVPEAR and the MILP formulation of the DAVINCI problem were solved using Gurobi v9.0 and the proposed heuristics implemented in Python 2.7. DAVINCI was implemented with the help of Multipliers following the guidelines provided in Fisher (1981a), Kwon, Changhyun (2016). Finally, all of the experiments were conducted on a machine equipped with an 8 core 2.3Ghz Intel Xeon Processor CPU and 62GB of memory running on an Xfce GNU/Linux 20.04.

5.6.4 Results

We evaluated DAVINCI and its policies against *NFV-PEAR* proposed in Miotto *et al.* (2019). *NFV-PEAR* is proposed as an adaptive placement and chaining model that tries to cause minimal reallocation of VNFs or network flows to deliver stable functioning of the SFCs. We implemented NFV-PEAR from scratch, and adapted its implementation to perform our experiments for a fair comparison with respect to its original description.

To start with, we discuss the results obtained for the first scenario (e.g., non sensitive requests), and then, due to page limitations and for more clarity, we provide a side-by-side compari-

son to highlight how much better/worse our multi-objective heuristics are compared to the abovementioned solutions for both scenarios.

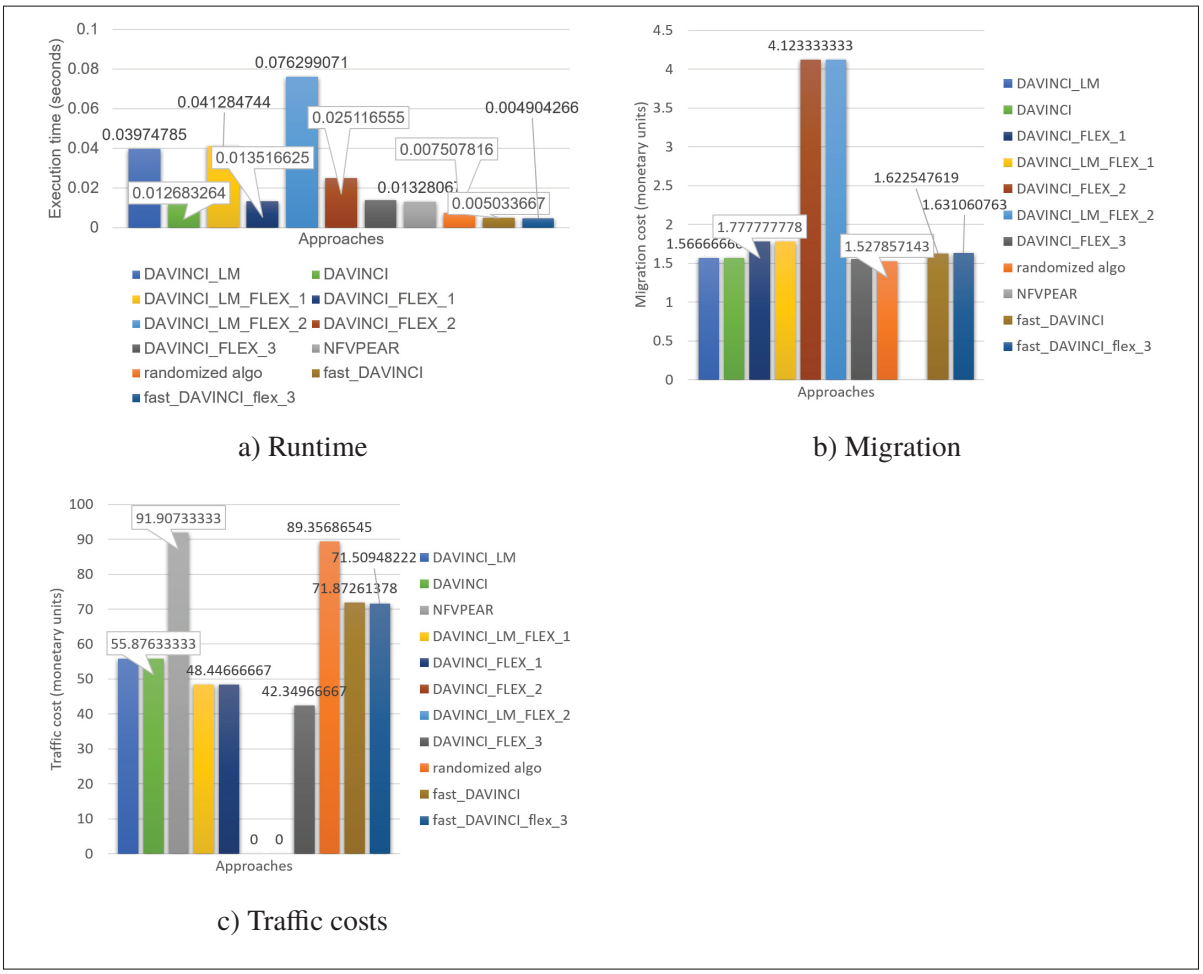


Figure 5.2 Runtime, migration and traffic costs

Figure 5.2a, 5.2b and 5.2c help us to understand the average adaptation time, the average migration cost per the number of impacted requests and the average traffic cost resulting from the different approaches, respectively. It can be clearly seen that fast_DAVINCI_flex_3 is the fastest among the different approaches. This is primarily due to its computational complexity, which is linear, followed by fast_DAVINCI. Another observation relates to NFV-PEAR which is faster than DAVINCI_LM, DAVINCI_LM_FLEX_1, DAVINCI_LM_FLEX_2, DAVINCI_FLEX_1, DAVINCI_FLEX_2 and DAVINCI_FLEX_3, respectively. The rationale behind this stems from the design of NFV-PEAR, which tries to cause minimal perturbations to a request with

respect to its initial deployment, thus keeping the location of VNFs and the paths connecting them unchanged unless the SLO is at risk of being violated. However, DAVINCI_LM_FLEX_-2 exhibits the highest runtime, becasue it employs the Lagrange Multiplier iterative-based technique to minimize the SLO violation penalty cost in addition to the optimization of the whole request following its adaptation strategy. Otherwise, the variants of DAVINCI exhibit similar execution time and are faster than those using Lagrange Multipliers. Moving to the migration cost, the randomized algorithm displays the lowest cost compared to that of DAVINCI, DAVINCI_LM and their variants, as well as fast_DAVINCI and fast_DAVINCI_flex_3. This is a consequence of fewer requests being adapted where VNFs required migrations compared to the other approaches. DAVINCI_FLEX_1 (resp. DAVINCI_LM_FLEX_1) shows a slightly higher migration cost compared to DAVINCI (resp. DAVINCI_LM), because the VNFs requiring vertical scaling are also migrated to optimize the cost functions. DAVINCI (resp. DAVINCI_LM), DAVINCI_FLEX_1 (resp. DAVINCI_LM_FLEX_1), DAVINCI_FLEX_2 (resp. DAVINCI_LM_FLEX_2), DAVINCI_FLEX_3, NFV-PEAR, the randomized algorithm, fast_DAVINCI and fast_DAVINCI_flex_3 migrated 100% (resp. 100%), 100%(resp. 100%), 100% (resp. 100%), 100%, 0%, 62.5%, 56.25% and 62.5% of the VNFs within the decision set, respectively.

Finally, by examining the traffic cost, we can see that DAVINCI_FLEX_2 (resp. DAVINCI_-LM_FLEX_2) achieves the lowest cost owing to the consolidation strategy that triggered additional VNF migrations. As opposed to the aforementioned approach, NFV-PEAR preserved the placement and chaining of the VNFs unchanged. fast_DAVINCI, fast_DAVINCI_flex_3, DAVINCI (resp. DAVINCI_LM), DAVINCI_FLEX_1 (resp. DAVINCI_LM_FLEX_1) and DAVINCI_FLEX_3 show lower traffic costs compared to the randomized algorithm, because they optimize the allocated traffic of the adapted requests. However, the randomized algorithm just readapts the requests blindly without optimizing the allocated traffic. Finally, DAVINCI_FLEX_-3, DAVINCI (resp. DAVINCI_LM) and DAVINCI_FLEX_1 (resp. DAVINCI_LM_FLEX_1) display close traffic costs, because they preserved the locations of the VNFs that do not require

adaptation, but optimized the paths with respect to the adapted VNFs in comparison with NFV-PEAR.

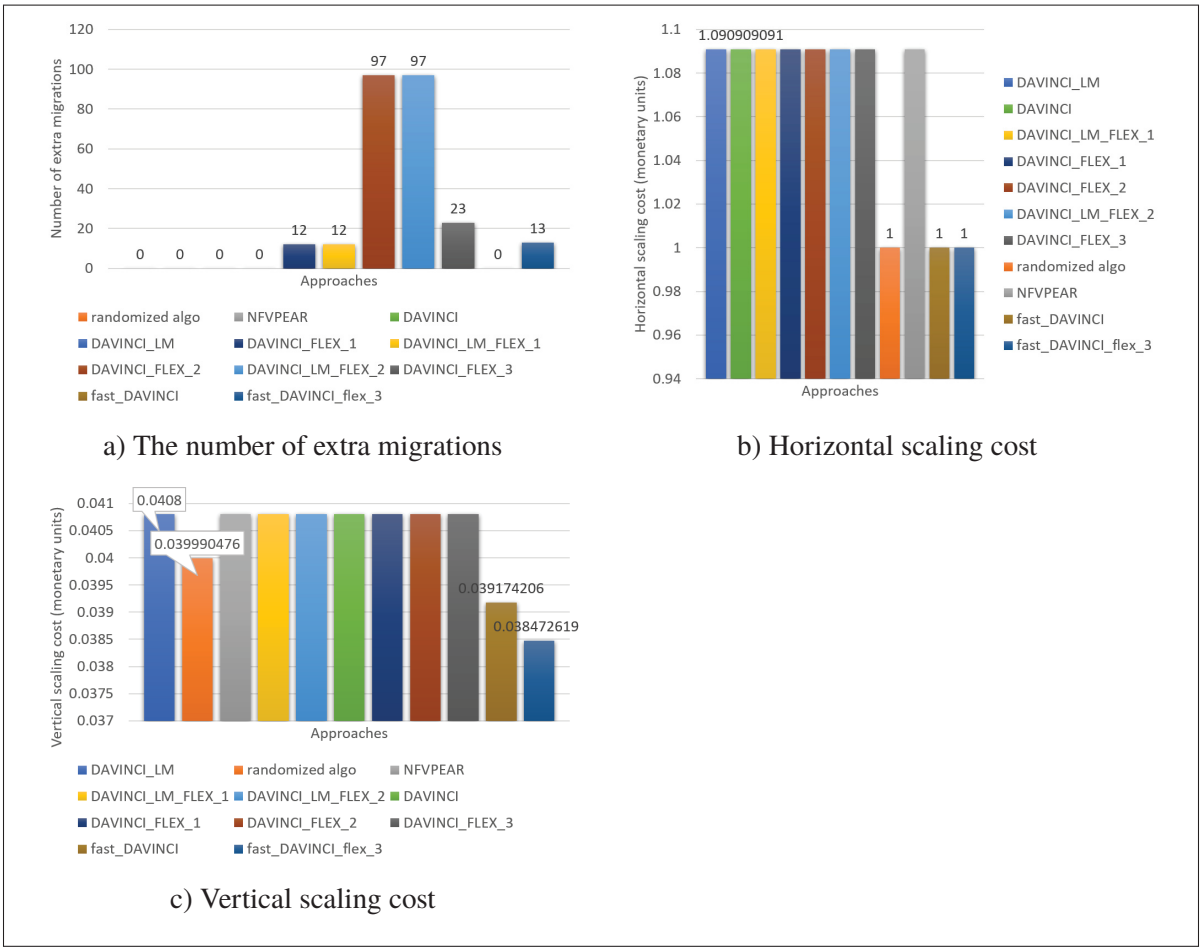


Figure 5.3 The number of extra migrations, horizontal and vertical scaling costs

Figure 5.3a, 5.3b and 5.3c report the number of extra migrations, horizontal and vertical scaling costs, respectively. Clearly, fast_DAVINCI, fast_DAVINCI_flex_3 and DAVINCI (resp. DAVINCI_LM), the randomized algorithm and NFV-PEAR exhibit the lowest number of additional migrations, bearing in mind that they migrate the VNFs with respect to the migration decision set. However, NFV-PEAR shows no extra migration, because no VNF is migrated at all, although the migration decision set contains a group of VNFs that must be relocated. In contrast, the number of migrated VNFs increases with DAVINCI_FLEX_1 (resp. DAVINCI_-LM_FLEX_1), DAVINCI_FLEX_2 (res. DAVINCI_LM_FLEX_2) and DAVINCI_FLEX_3.

This is because in addition to the VNFs that require migration with respect to the migration decision set, VNFs requiring vertical scaling are also migrated to optimize the cost functions. DAVINCI_FLEX_3 is identical to DAVINCI_FLEX_1, except that the VNFs requiring horizontal scaling are also migrated. For DAVINCI_FLEX_2 (resp. DAVINCI_LM_FLEX_2), one can observe that the results are consistent with those shown in figure 5.2b in terms of migration cost. In terms of horizontal scaling cost, we can see that the randomized algorithm displays the lowest cost compared to that of DAVINCI (resp. DAVINCI_LM) and its variants, along with NFV-PEAR. This is because the randomized algorithm adapted fewer VNFs with respect to the horizontal scaling decision set. However, the costs are the same for NFV-PEAR, DAVINCI (resp. DAVINCI_LM) and its variants, because they adapted all of the VNFs requiring horizontal scaling. fast_DAVINCI and fast_DAVINCI_flex_3 failed to adapt the requests where VNFs require horizontal scaling due to their inability to explore alternative areas of the search space in order to deploy them. Moving to the vertical scaling cost, fast_DAVINCI_flex_3 displays the lowest cost compared to the other approaches. As is the case with horizontal scaling, it was not able to find hosts with sufficient residual resources to relocate the adapted VNFs. However, the costs are the same for NFV-PEAR, DAVINCI (resp. DAVINCI_LM) and its variants, because they adapted all of the VNFs with respect to the vertical scaling decision set. fast_DAVINCI displays marginally higher vertical scaling cost compared to fast_DAVINCI_3, because it adapted one more request.

DAVINCI (resp. DAVINCI_LM), DAVINCI_FLEX_1 (resp. DAVINCI_LM_FLEX_1), DAVINCI_FLEX_2 (resp. DAVINCI_LM_FLEX_2), DAVINCI_FLEX_3, NFV-PEAR, the randomized algorithm, fast_DAVINCI and fast_DAVINCI_flex_3 horizontally scaled 100% (resp. 100%), 100%(resp. 100%), 100% (resp. 100%), 100%, 100%, 41.67%, 41.67% and 50% of the VNFs within the decision set, respectively. In terms of vertical scaling, DAVINCI (resp. DAVINCI_LM), DAVINCI_FLEX_1 (resp. DAVINCI_LM_FLEX_1), DAVINCI_FLEX_2 (resp. DAVINCI_LM_FLEX_2), DAVINCI_FLEX_3, NFV-PEAR, the randomized algorithm, fast_DAVINCI and fast_DAVINCI_flex_3 horizontally scaled 100% (resp. 100%), 100%(resp.

100%), 100% (resp. 100%), 100%, 100%, 75%, 75% and 66.67% of the VNFs within the decision set, respectively.

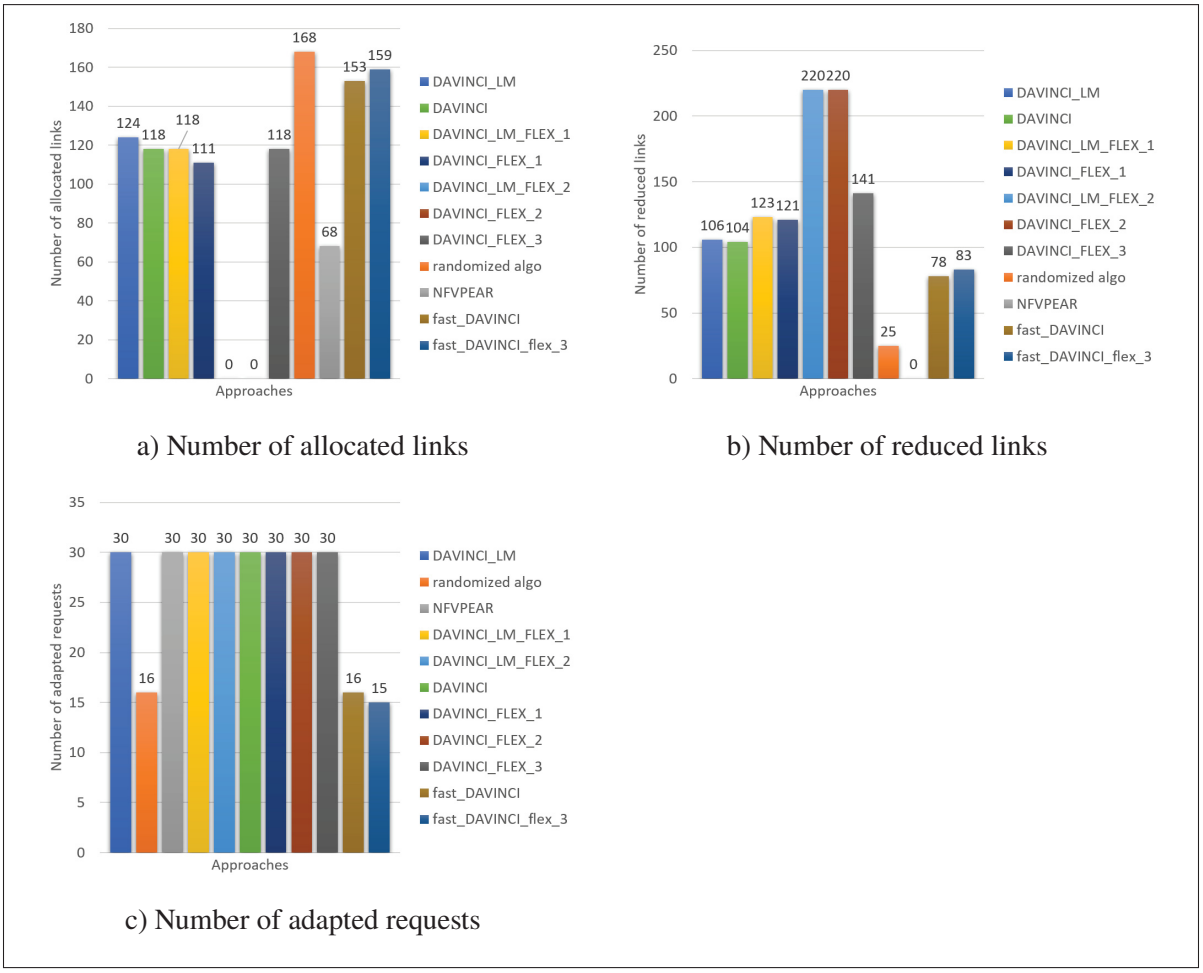


Figure 5.4 Number of allocated links, number of reduced links and the number of adapted requests

Now let us examine the number of allocated and reduced links for all of the adapted requests. As we can see from figure 5.4a, DAVINCI_FLEX_2 (resp. DAVINCI_LM_FLEX_2) displays the lowest number of allocated links after adaptation, because it consolidated the VNFs. NFV-PEAR shows a lower number of allocated links compared to DAVINCI (resp. DAVINCI_LM), DAVINCI_FLEX_1 (resp. DAVINCI_LM_FLEX_1) and DAVINCI_FLEX_3, which represents the total number of allocated links for all of the previously deployed requests before adaptation. fast_DAVINCI and fast_DAVINCI_flex_3 exhibit higher numbers of allocated links in the new

configuration, because they optimized all of the paths of the adapted requests. The randomized algorithm displays higher numbers of allocated links in the new configuration, because no path optimizations were performed when adapting the VNFs. DAVINCI_FLEX_3 displays a higher number of allocated links in contrast to DAVINCI_FLEX_1 in the new configuration compared to the previous configuration due to the migration, horizontal and vertical scaling decisions. However, DAVINCI_LM displays s higher number of newly allocated links due to the minimization of SLO leading to alternative paths in the new configuration of the adapted requests. Following this analysis, DAVINCI_FLEX_2 (resp. DAVINCI_LM_FLEX_2) displays the highest number of reduced links, because all of the VNFs were migrated and consolidated within the same hosts, followed by DAVINCI_FLEX_3, DAVINCI_FLEX_1 (resp. DAVINCI_-LM_FLEX_1) and DAVINCI (res. DAVINCI_LM), fast_DAVINCI, fast_DAVINCI_flex_3 and DAVINCI. Moreover, DAVINCI_FLEX_1 reduced the length of the paths from the previous configuration for the adapted requests in the new configuration by optimizing the traffic costs that lead to variant paths compared to the randomized algorithm. In contrast to these approaches, NFV-PEAR preserved the initial deployment of the requests. Finally, the randomized algorithm, fast_DAVINCI and fast_DAVINCI_flex_3 achieved a lower ratio of adapted requests of 53.33%, 53.33% and 50%, respectively, compared to the other approaches, which accomplished a proportion of 100%. Recall that NFV-PEAR and the randomized algorithm adapted only 0% and 62.5% of the VNFs that required migration from the decision set. The vision of NFV-PEAR ended up migrating other VNFs to enforce the SLO instead of violating it.

To sum it up, DAVINCI (resp. DAVINCI_LM) and its variants and NFV-PEAR use the residual resources of the infrastructure efficiently based on the total cost, which includes migration, vertical and horizontal scaling and the allocated traffic optimization. We observe similar trends with respect to the second scenario, except that all of the approaches undergo a sharp increase in terms of runtime. This has to do with the sizes of the infrastructure and SFCs. Another observation relates to the traffic cost, which almost doubles because we have more VNFs and path allocations compared to the first scenario, resulting in the impacts on the total cost.

Scenario (1/2)	DAVINCI/Randomized algorithm/NFVPEAR/DAVINCI_FLEX_1/DAVINCI_FLEX_2/DAVINCI_FLEX_3/fast_DAVINCI/fast_DAVINCI_flex_3									
	Runtime (sec)	SLO violation costs (mu)	Migration costs (mu)	Traffic costs (mu)	VS costs (mu)	HS costs (mu)	Total costs (mu)	MD costs (mu)	# of successfully adapted requests (%)	Number of rejected requests
Non sensitive requests	0.0127/ 0.0075/ 0.0133/ 0.0135/ 0.0251/ 0.014/ 0.005/ 0.0049	0 / 0 / 0 /0 / 0 / 0 / 0 / 0	1.56 / 1.52 / 0 / 1.77 / 4.12 / 1.55 / 1.62 / 1.63	55.87 / 89.35 / 91.90 / 48.44 / 0 / 42.35 / 71.87/ 71.50	0.0408 / 0.0399/ 0.0408 / 0.0408 / 0.0408 / 0.03917 / 0.03847	1.09 / 1 / 1.09 / 1.09 / 1.09 / 1.09 / 1 / 1	58.57 / 91.92 / 93.04 / 51.36 / 5.26 / 45.03 / 74.53 / 74.18	0 / 0 / 0 / 0 / 0 / 0 / 0 / 0	100 / 53.33 / 100 / 100 / 100 / 100 / 53.33 / 50	0 / 14 / 0 / 0 / 0 / 0 / 14 / 15
Sensitive requests	0.0127 / 0.0077 / 0.0249 / 0.0127 / 0.022 / 0.013 / 0.0049 / 0.0048	0 .011/ 0 / 0 / 0.01 / 0 / 0.01 / 0 / 0	1.257 / 1.065 / 1.3375 / 1.4818 / 4.0467 / 1.128 / 1.2098 / 1.535	63.47 / 91.43 / 84.857 / 59.33 / 0 / 52.41 / 73.97 / 71.63	0.03825/ 0.03822/ 0.03825/ 0.03825/ 0.03825/ 0.03825 / 0.0366 / 0.0361	1.22 / 2 / 1.22 / 1.22 / 1.22 / 1.22 / 2 / 2	66.00 / 94.53 / 87.455 / 62.088 / 5.30 / 54.81 / 77.22 / 75.20	0 / 0 / 0 / 0 / 0 / 0 / 0 / 0	100 / 56.67 / 100 / 100 / 100 / 100 / 56.67 / 50	0 / 13 / 0 / 0 / 0 / 0 / 13 / 15
Highly sensitive requests	0.0114 / 0.0077 / 0.01881 / 0.01239 / 0.02612 / 0.01277 / 0.005 / 0.0048	0.0342/ 0.0142 / 0 / 0.03453 / 0 / 0.03675 / 0 / 0	1.39 / 1.36 / 3.66 / 1.495 / 3.91 / 1.237 / 1.44 / 1.61	50.33 / 75.013 / 16.878 / 42.60 / 0.12 / 41.08 / 61.41 / 60.41	0.037/ 0.039 / 0.037 / 0.037 / 0.037 / 0.037 / 0.0389 / 0.0391	1 / 1 / 1.1667 / 1 / 1.1667 / 1 / 1 / 1	52.79 / 77.43 / 21.74 / 45.17 / 5.24 / 43.39 / 63.89 / 63.06	0 / 0 / 0 / 0 / 0 / 0 / 0 / 0	90 / 46.67 / 100 / 90 / 100 / 90 / 46.67 / 50	3 / 16 / 0 / 3 / 0 / 3 / 16 / 15

Figure 5.5 Performance metrics for the different approaches (Scenario 1)

Scenario (2/2)	DAVINCI/Randomized algorithm/NFVPEAR/DAVINCI_FLEX_1/DAVINCI_FLEX_2/DAVINCI_FLEX_3/fast_DAVINCI/fast_DAVINCI_flex_3									
	Runtime (sec)	SLO violation costs (mu)	Migration costs (mu)	Traffic costs (mu)	VS costs (mu)	HS costs (mu)	Total costs (mu)	MD costs (mu)	# of adapted requests (%)	Number of rejected requests
Non sensitive requests	4.103 /6.24 /3.017 /4.48 /12.22 /4.642 / 1.439 / 1.39	0 / 0 / 0 / 0 / 0 / 0 / 0 / 0	1.56 / 1.576 / 0 / 1.77 / 4.346 / 1.5857 / 1.54 / 1.75	114.29 / 227.30 / 406.21 / 104.25 / 0 / 95.056 / 136.58 / 138.20	0.0408 / 0.0408/ 0.0408 / 0.0408 / 0.0408 / 0.0408 / 0.0402 / 0.0406	1.09 / 2 / 1.09 / 1.09 / 1.09 / 1.09 / 2 / 2	116.98 / 230.91 / 407.34 / 107.16 / 5.47 / 97.77 / 140.17 / 142	0 / 0 / 0 / 0 / 0 / 0 / 0 / 0	100 / 100 / 100 / 100 / 100 / 100 / 100 / 100	0 /0 /0 /0 /0 / 0/ 0 / 0
Sensitive requests	3.958 / 7.1535 / 3.123 / 4.272 / 11.93 / 5.053 / 1.44 / 1.36	0 / 0 / 0 / 0 / 0 / 0 / 0 / 0	1.257 / 1.2143 / 0 / 1.423 / 4.317 / 1.216 / 1.23 / 1.68	122.1 / 228.13 / 494.2 / 113.2 / 0 / 100.9 / 138.04 / 137.89	0.0383/ 0.0382/ 0.0382/ 0.0382 / 0.0382 / 0.0382 / 0.0383 / 0.0383	1.22 / 2 / 1.22 / 1.22 / 1.22 / 1.22 / 2 / 2	124.6 / 231.4 / 495.5 / 115.8 / 5.577/ 103.4 / 141.31 / 141.61	0 / 0 / 0 / 0 / 0 / 0 / 0 / 0	100 / 96.67 / 100 / 100 / 100 /100 / 96.67 / 100	0 / 1 / 0 / 0/ 0/ 0/ 1 / 0
Highly sensitive requests	3.815 / 6.0177 / 2.99 / 4.009 / 10.96 / 4.442 / 1.45 / 1.37	0 / 0 / 0 / 0 / 0 / 0 / 0 / 0	1.354 / 1.3538 / 0 / 1.467 / 4.2 / 1.334 / 1.35 / 1.54	111.097 / 221.639 / 444.71 / 100.915 / 0 / 93.44 / 129.64 / 130.56	0.0370 /0.0370 / 0.0370 / 0.0370 / 0.0370 / 0.0367 / 0.0370	1.166 / 2/ 1.16 / 1.16/ 1.16 / 1.16 / 2 / 2	113.65 / 225.03 / 445.91 / 103.58 / 5.40 / 95.98 / 133.03 / 134.14	0 / 0 / 0 / 0 / 0 / 0 / 0 / 0	100 / 100 / 100 / 100 / 100 / 100 / 100 / 100	0 / 0 / 0 / 0 / 0 / 0 / 0 / 0

Figure 5.6 Performance metrics for the different approaches (Scenario 2)

Scenario		NFVPEAR vs DAVINCI_FLEX_2		NFVPEAR vs fast_DAVINCI_flex_3	
		Total cost ratio (NFVPEAR / DAVINCI_FLEX_2)	Execution time ratio (DAVINCI_FLEX_2 / NFVPEAR)	Total cost ratio (NFVPEAR / fast_DAVINCI_flex_3)	Execution time ratio (fast_DAVINCI_FLEX_3 / NFVPEAR)
#1	Non sensitive requests	DAVINCI_FLEX_2 is almost 17.7 times less expensive than NFVPEAR	DAVINCI_FLEX_2 is 1.89 order of magnitude slower than NFVPEAR	fast_DAVINCI_flex_3 is almost 1.25 times less expensive than NFVPEAR	Fast_DAVINCI_flex_3 is 2.70 order of magnitude faster than NFVPEAR
	Sensitive requests	DAVINCI_FLEX_2 is almost 16.5 times less expensive than DAVINCI_FLEX_2	DAVINCI_FLEX_2 is 1.13 order of magnitude slower than NFVPEAR	Fast_DAVINCI_flex_3 is almost 1.2 times less expensive than fast_DAVINCI_flex_3	Fast_DAVINCI_flex_3 is 5.13 order of magnitude faster than NFVPEAR
	Highly sensitive requests	DAVINCI_FLEX_2 is almost 4.2 times less expensive than DAVINCI_FLEX_2	DAVINCI_FLEX_2 is 1.38 order of magnitude slower than NFVPEAR	Fast_DAVINCI_flex_3 is almost 2.9 times more expensive than fast_DAVINCI_flex_3	Fast_DAVINCI_flex_3 is 3.90 order of magnitude faster than NFVPEAR
#2	Non sensitive requests	DAVINCI_FLEX_2 almost 74.5 times less expensive than NFVPEAR	DAVINCI_FLEX_2 is 4.05 order of magnitude slower than NFVPEAR	Fast_DAVINCI_flex_3 is almost 2.86 times less expensive than fast_DAVINCI_flex_3	Fast_DAVINCI_flex_3 is 2.17 order of magnitude faster than NFVPEAR
	Sensitive requests	DAVINCI_FLEX_2 almost 88.8 times less expensive than NFVPEAR	DAVINCI_FLEX_2 is 3.82 order of magnitude slower than NFVPEAR	Fast_DAVINCI_flex_3 is almost 3.5 times less expensive than fast_DAVINCI_flex_3	Fast_DAVINCI_flex_3 is 2.29 order of magnitude faster than NFVPEAR
	Highly sensitive requests	DAVINCI_FLEX_2 almost 82.6 times less expensive than NFVPEAR	DAVINCI_FLEX_2 is 3.65 order of magnitude slower than NFVPEAR	Fast_DAVINCI_flex_3 is almost 3.3 times less expensive than fast_DAVINCI_flex_3	Fast_DAVINCI_flex_3 is 2.18 order of magnitude faster than NFVPEAR

Figure 5.7 Comparative analysis (DAVINCI_FLEX_2/fast_DAVINCI_flex_3 with NFV-PEAR) (runtime and total cost)

However, the number of rejected requests is dramatically reduced, because we have more available resources to afford the adaptation of the virtual network services. Finally, figures 5.5, 5.6 and 5.7 illustrate the performance of the different approaches achieved with respect to the types and sizes of the SFCs and the considered network infrastructures. It can be clearly seen that DAVINCI and its variants outperform NFV-PEAR in terms of total cost (figure 5.7).

5.7 Conclusion

In this paper, we proposed an online and dynamic adaptation of evolvable SFC models and studied the impact of decision adaptation policies that comprise elasticity mechanisms and migration. The problem that we addressed in this paper is of paramount importance for network and service providers, because it raises several challenges that must be considered before and while adapting virtual network services. We have investigated different decision adaptation policies that make use of elasticity mechanisms and migration technique and evaluated their costs and overheads. The experimental study shows that, compared to NFV-PEAR, DAVINCI displays lower cost and overhead with respect to the different decision adaptation policies. However, the experimental study demonstrates that different adaptation policies result in different adaptation costs and there is no one-size-fits-all solution.

We hope that DAVINCI will establish the basis for further research in policy-based dynamic adaptation and SFC orchestration. One interesting future research direction would be to investigate the impacts of the decision adaptation policies on the SFCs from a system perspective, using a real test-bed, which would certainly provide us with relevant insights in order to fine-tune the proposed mathematical model and the heuristics.

Acknowledgments

This work has been supported by Ericsson Canada and the Natural Sciences and Engineering Research Council of Canada (NSERC).

CONCLUSION AND RECOMMENDATIONS

Apart from the advantages and benefits that virtualization brings to service providers and network operators, there are still many challenges to overcome before the transition to full virtualization and network services deployment. The major challenge that we take up within the framework of this thesis remains that linked to the algorithmic complexity of the proposed approaches, whether for the deployment of service chains or even their dynamic adaptations. Indeed, for a large-scale deployment with architecture including at the same time the cloud, the edge, and the internet of things, it is essential to develop approaches that at the same time exhibit low algorithmic complexity and are scalable for a better tradeoff.

As part of this thesis, we have dealt with the dynamic placement of service chains in chapter 2. Most of the existing approaches which propose to solve this problem consider linear service chains with limited size (e.g., 3 to 5 VNFs). Therefore, the end-to-end delay is calculated based on the topology's linearity for some approaches, while others ignore it entirely to simplify the problem. For our part, we considered the end-to-end delay between two pairs of VNFs, i.e., at the virtual link-level precisely, to consider complex topologies. Therefore, our proposal makes it possible to deal with service chains of different sizes and topologies thanks to our physical algorithm, i.e., cultural genetic algorithm, by proposing a custom genotype encoding to this problem. In chapter 3, before the placement and chaining of the VNFs, we have presented a VNF selection mechanism that, given a pool of VNFs supplied by different vendors, chooses the VNFs that fit the requirements of the client's in terms of key performance indicators for delay-sensitive services. Once the VNFs are selected to compose the service function chain, the next step is to deploy them using a chemical reaction optimization approach.

To reduce the execution time for large-scale network architectures, we have proposed network partitioning techniques, in chapter 4, to build on-demand partitions while taking into account the requirements of the service chains in terms of computing and network resources (e.g., CPU,

Memory, and Bandwidth) in addition to the QoS (e.g., end-to-end delay) constraints. In this way, we will be able to reduce the execution time of our approach to placing and chaining VNFs and guarantee that a solution will be found which meets the requirements of the service chain's requests. This is not the case if we do not split the network according to the service chains' needs. Indeed, since an evolutionary algorithm is stochastic by nature, it may not find a feasible solution on the first attempt because the size of the network is huge, and the possibilities for placing VNFs are immense. Therefore, several attempts might be necessary. However, this situation can be avoided with on-demand partitioning techniques that reduce the search space significantly.

Regarding the dynamic adaptation of service chains, we have proposed, in chapter 5, a model and a suite of heuristics whose complexity is lower to react quickly to various disturbances (e.g., traffic fluctuations) to minimize the costs of penalties (e.g., migration downtime and end-to-end delay). Besides, the elasticity mechanisms (e.g., migration procedure, vertical and horizontal scaling) are included as binary decisions made for each of the VNFs in the service chain. Several adaptation policies are studied and analyzed with several performance metrics to enable decision-makers to make decisions and choose the best way to adapt. Our approach can work in tandem with prediction-based methods to decide what action to take for any VNF to see its impact across the entire service chain to reduce cost penalties. Moreover, it can be executed proactively or just invoked reactively following a set of trigger events (e.g., migration) to adapt the service function chain efficiently with the least cost.

We sincerely believe that our humble contributions to this thesis are an addition to the existing literature. We hope that our solutions developed in this thesis may give rise to other research avenues and a starting point for doctoral students who will continue their research on the same topic. As a prospect for future research, we plan to:

Develop approaches by exploiting time-efficient data structures : to reduce the complexity of VNF placement and chaining approaches and their adaptations. The idea is to manipulate data structures such as Union-Find to minimize the overall algorithmic complexity by avoiding using shortest path computation algorithms to interconnect VNFs. As a result, we will have very low algorithmic complexity solutions that can scale very well with problem instances (e.g., medium-scale and large-scale infrastructure's size) contrary to the literature's approaches. A master's student is working on this track, and the results are auspicious as we can guarantee almost linear complexity. The proposed approach is implemented in C++ with more than 1200 lines of code.

To develop an end-to-end delay computation model : for complex service chains using the Queueing theory. Then, integrate it into the decision-making process for the placement and chaining of the VNFs and their dynamic adaptation.

To investigate the estimation of migration downtime in a collaborative manner : existing procedures ignore the connectivity between the VNFs/VMs when computing the migration downtime. The calculation of the migration downtime needs to be achieved collaboratively to involve the VNFs to try to estimate the perceived downtime concerning the migrated VNF to which they are interconnected. Therefore, an efficient distributed mechanism is required to perform such computation.

Test bed evaluation of the proposed approaches : as most of our contributions have been evaluated against optimal approaches using Gurobi as a mathematical solver, we intend to implement them into real testbeds and assess their performance in such an entire network infrastructure. We want to evaluate our proposed approaches using network simulators such as Omnet++, Mininet, and NS-3.

As a reflection from my PhD journey : our last contribution enabled us to understand profoundly the problems we adressed in this thesis by proposing a differnt approach that is time-efficient. It would be of great interest to figure out how to tackle such problems from another radically different perspective. Perhaps some ways of thinking and mathematical tools are genuinely efficient because when we use them, they make such problems easily solvable with more time-efficient algorithms.

BIBLIOGRAPHY

Abdelsalam, M., Krishnan, R. & Sandhu, R. (2017). Clustering-Based IaaS Cloud Monitoring. *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pp. 672–679. doi: 10.1109/CLOUD.2017.90.

Addis, B., Belabed, D., Bouet, M. & Secci, S. (2015). Virtual network functions placement and routing optimization. *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*, pp. 171–177. doi: 10.1109/CloudNet.2015.7335301.

Addis, B., Gao, M. & Carello, G. (2018). On the complexity of a Virtual Network Function Placement and Routing problem. *Electronic Notes in Discrete Mathematics*, 69, 197–204. doi: 10.1016/j.endm.2018.07.026.

Al-Dhuraibi, Y., Paraiso, F., Djarallah, N. & Merle, P. (2018). Elasticity in Cloud Computing: State of the Art and Research Challenges. *IEEE Transactions on Services Computing*, 11(2), 430–447. doi: 10.1109/TSC.2017.2711009.

Alleg, A., Ahmed, T., Mosbah, M., Riggio, R. & Boutaba, R. (2017). Delay-aware VNF placement and chaining based on a flexible resource allocation approach. *2017 13th International Conference on Network and Service Management (CNSM)*, pp. 1–7. doi: 10.23919/CNSM.2017.8255993.

ApS, M. Mosek ApS. Retrieved from: https://www.mosek.com/.

Asgarian, M., Mirjalily, G. & Luo, Z.-Q. (2020). Embedding Multicast Service Function Chains in NFV-Enabled Networks. *IEEE Communications Letters*, 1–1. doi: 10.1109/LCOMM.2020.3044894.

Aylani, A. & Goyal, N. (2017). Community detection in social network based on useras social activities. *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pp. 625–628. doi: 10.1109/I-SMAC.2017.8058254.

Ayoubi, S., Chowdhury, S. R. & Boutaba, R. (2018). Breaking Service Function Chains with Khaleesi. *2018 IFIP Networking Conference (IFIP Networking) and Workshops*, pp. 64–72. doi: 10.23919/IFIPNetworking.2018.8697025.

Bari, M. F., Chowdhury, S. R., Ahmed, R. & Boutaba, R. (2015). On orchestrating virtual network functions. *2015 11th International Conference on Network and Service Management (CNSM)*, pp. 50–56. doi: 10.1109/CNSM.2015.7367338.

Baroni, A., Conte, A., Patrignani, M. & Ruggieri, S. (2017). Efficiently Clustering Very Large Attributed Graphs. *2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pp. 369–376.

Bechikh, S., Chaabani, A. & Said, L. B. (2015). An Efficient Chemical Reaction Optimization Algorithm for Multiobjective Optimization. *IEEE Transactions on Cybernetics*, 45(10), 2051–2064. doi: 10.1109/TCYB.2014.2363878.

Bishop, C. M. (2006). Pattern recognition and machine learning. ISBN: 9781493938438 9780387310732 Publisher: Springer, Retrieved from: https://cds.cern.ch/record/998831.

Bonfim, M. S., Dias, K. L. & Fernandes, S. F. L. (2019a). Integrated NFV/SDN Architectures: A Systematic Literature Review. *ACM Computing Surveys*, 51(6), 114:1–114:39. doi: 10.1145/3172866.

Bonfim, M. S., Dias, K. L. & Fernandes, S. F. L. (2019b). Integrated NFV/SDN Architectures: A Systematic Literature Review. *ACM Computing Surveys*, 51(6), 114:1–114:39. doi: 10.1145/3172866.

Bothorel, C., Cruz, J. D., Magnani, M. & Micenková, B. (2015). Clustering attributed graphs: Models, measures and methods. *Network Science*, 3(3), 408–444. doi: 10.1017/nws.2015.9. Publisher: Cambridge University Press.

Brownlee, J. (2011). *Clever Algorithms: Nature-Inspired Programming Recipes* (ed. 1st). Lulu.com.

Carpio, F., Jukan, A. & Pries, R. (2018). Balancing the migration of virtual network functions with replications in data centers. *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–8. doi: 10.1109/NOMS.2018.8406275.

Casado, M., Freedman, M. J., Pettit, J., Luo, J., McKeown, N. & Shenker, S. (2007). Ethane: taking control of the enterprise. *ACM SIGCOMM Computer Communication Review*, 37(4), 1–12. doi: 10.1145/1282427.1282382.

Cerroni, W. & Callegati, F. (2014). Live migration of virtual network functions in cloud-based edge networks. *2014 IEEE International Conference on Communications (ICC)*, pp. 2963–2968. doi: 10.1109/ICC.2014.6883775.

Chavan, V. & Kaveri, P. R. (2014). Clustered virtual machines for higher availability of resources with improved scalability in cloud computing. *2014 First International Conference on Networks Soft Computing (ICNSC2014)*, pp. 221–225. doi: 10.1109/CNSC.2014.6906707.

Chen, C., Du, Y., Chen, S. & Wang, W. (2018). Partitioning and Placing Virtual Machine Clusters on Cloud Environment. *2018 1st International Cognitive Cities Conference (IC3)*, pp. 268–270. doi: 10.1109/IC3.2018.000-2.

Chen, S.-J., Chen, C.-C., Lu, H.-L. & Wang, W.-J. (2017). Efficient resource provisioning for virtual clusters on the cloud. *International Journal of Services Technology and Management*, 23(1-2), 52–63. doi: 10.1504/IJSTM.2017.081876. Publisher: Inderscience Publishers.

Cheng, H., Zhou, Y. & Yu, J. X. (2011). Clustering Large Attributed Graphs: A Balance between Structural and Attribute Similarities. *ACM Transactions on Knowledge Discovery from Data*, 5(2), 12:1–12:33. doi: 10.1145/1921632.1921638.

Cho, D., Taheri, J., Zomaya, A. Y. & Bouvry, P. (2017). Real-Time Virtual Network Function (VNF) Migration toward Low Network Latency in Cloud Environments. *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pp. 798–801. doi: 10.1109/CLOUD.2017.118.

Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I. & Warfield, A. (2005). Live migration of virtual machines. *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2*, (NSDI'05), 273–286.

Clauset, A., Newman, M. E. J. & Moore, C. (2004). Finding community structure in very large networks. *Physical Review E*, 70(6), 066111. doi: 10.1103/PhysRevE.70.066111. Publisher: American Physical Society.

Cormen,Thomas H and Leiserson, C. E. a. R. R. L. a. S. C. (2009). *Introduction to Algorithms, 3rd-edition*. MIT Press and McGraw-Hill. Retrieved from: https://mitpress.mit.edu/books/introduction-algorithms-third-edition.

Dahmen-Lhuissier, S. ETSI - Open Source Mano | Open Source Solutions | Mano NFV. Retrieved from: https://www.etsi.org/technologies/open-source-mano?jjj=1612367204152.

DataSwitchWorks. Brocade vEPC. Retrieved from: https://www.dataswitchworks.com/vEPC.asp.

DataSwitchWorks. Brocade Virtual Router. Retrieved from: https://www.dataswitchworks.com/vRouter.asp.

David Hadka. (2015). Platypus. original-date: 2015-10-09T19:19:17Z, Retrieved from: https://github.com/Project-Platypus/Platypus.

Deb, K., Pratap, A., Agarwal, S. & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2), 182–197. doi: 10.1109/4235.996017.

Deric, N., Varasteh, A., Basta, A., Blenk, A., Pries, R., Jarschel, M. & Kellerer, W. (2019). Coupling VNF Orchestration and SDN Virtual Network Reconfiguration. *2019 International*

Conference on Networked Systems (NetSys), pp. 1–3. doi: 10.1109/NetSys.2019.8854520.

Deza, M. M. & Deza, E. (2009). Encyclopedia of Distances. In Deza, E. & Deza, M. M. (Eds.), *Encyclopedia of Distances* (pp. 1–583). Berlin, Heidelberg: Springer. doi: 10.1007/978-3-642-00234-2_1.

Dominicini, C. K., Vassoler, G. L., Valentim, R., Villaca, R. S., Ribeiro, M. R. N., Martinello, M. & Zambon, E. (2020). KeySFC: Traffic steering using strict source routing for dynamic and efficient network orchestration. *Computer Networks*, 167, 106975. doi: 10.1016/j.comnet.2019.106975.

D'Oro, S., Galluccio, L., Palazzo, S. & Schembra, G. (2017). A marketplace as a scalable solution to the orchestration problem in SDN/NFV networks. *2017 IEEE Conference on Network Softwarization (NetSoft)*, pp. 1–5. doi: 10.1109/NETSOFT.2017.8004224.

Draxler, S., Karl, H. & Mann, Z. A. (2018). JASPER: Joint Optimization of Scaling, Placement, and Routing of Virtual Network Services. *IEEE Transactions on Network and Service Management*, 15(3), 946–960. doi: 10.1109/TNSM.2018.2846572.

Duan, J., Wu, C., Le, F., Liu, A. X. & Peng, Y. (2017). Dynamic Scaling of Virtualized, Distributed Service Chains: A Case Study of IMS. *IEEE Journal on Selected Areas in Communications*, 35(11), 2501–2511. doi: 10.1109/JSAC.2017.2760188.

Duan, Q. (2018). Modeling and Performance Analysis for Service Function Chaining in the SDN/NFV Architecture. *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pp. 476–481. doi: 10.1109/NETSOFT.2018.8460068.

Duong-Ba, T. H., Nguyen, T., Bose, B. & Tran, T. T. (2018). A Dynamic Virtual Machine Placement and Migration Scheme For Data Centers. *IEEE Transactions on Services Computing*, 1–1. doi: 10.1109/TSC.2018.2817208.

D'Oro, S., Galluccio, L., Palazzo, S. & Schembra, G. (2017a). Exploiting Congestion Games to Achieve Distributed Service Chaining in NFV Networks. *IEEE Journal on Selected Areas in Communications*, 35(2), 407–420. doi: 10.1109/JSAC.2017.2659298.

D'Oro, S., Palazzo, S. & Schembra, G. (2017b). Orchestrating Softwarized Networks with a Marketplace Approach. *Procedia Computer Science*, 110, 352–360. doi: 10.1016/j.procs.2017.06.077.

El Mensoum, I., Wahab, O. A., Kara, N. & Edstrom, C. (2020). MuSC: A multi-stage service chains embedding approach. *Journal of Network and Computer Applications*, 159, 102593. doi: 10.1016/j.jnca.2020.102593.

Eramo, V., Ammar, M. & Lavacca, F. G. (2017a). Migration Energy Aware Reconfigurations of Virtual Network Function Instances in NFV Architectures. *IEEE Access*, 5, 4927–4938. doi: 10.1109/ACCESS.2017.2685437.

Eramo, V., Miucci, E., Ammar, M. & Lavacca, F. G. (2017b). An Approach for Service Function Chain Routing and Virtual Function Network Instance Migration in Network Function Virtualization Architectures. *IEEE/ACM Transactions on Networking*, 25(4), 2008–2025. doi: 10.1109/TNET.2017.2668470.

Ester, M., Kriegel, H.-P., Sander, J. & Xu, X. A density-based algorithm for discovering clusters in large spatial databases with noise.

ETSI. ETSI. Retrieved from: https://www.etsi.org/technologies/nfv.

ETSI-5G. ETSI. Retrieved from: https://www.etsi.org/technologies/5G.

ETSI-NS. ETSI. Retrieved from: https://www.etsi.org/deliver/etsi_gr/NFV-EVE/001_099/012/03.01.01_60/gr_NFV-EVE012v030101p.pdf.

Falkenauer, E. (1996). A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2(1), 5–30. doi: 10.1007/BF00226291.

Fangxin Wang, Ruilin Ling, Zhu, J. & Li, D. (2015). Bandwidth guaranteed virtual network function placement and scaling in datacenter networks. *2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC)*, pp. 1–8. doi: 10.1109/PCCC.2015.7410276.

Fischer, A., Bhamare, D. & Kassler, A. (2019). On the Construction of Optimal Embedding Problems for Delay-Sensitive Service Function Chains. *2019 28th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–10. doi: 10.1109/ICCCN.2019.8847151.

Fisher, M. L. (1981a). The Lagrangian Relaxation Method for Solving Integer Programming Problems. *Management Science*, 27(1), 1–18. doi: 10.1287/mnsc.27.1.1. Publisher: INFORMS.

Fisher, M. L. (1981b). The Lagrangian Relaxation Method for Solving Integer Programming Problems. *Management Science*, 27(1), 1–18. doi: 10.1287/mnsc.27.1.1. Publisher: INFORMS.

Gandhi, A., Harchol-Balter, M., Das, R. & Lefurgy, C. (2009). Optimal power allocation in server farms. *ACM SIGMETRICS Performance Evaluation Review*, 37(1), 157–168. doi: 10.1145/2492101.1555368.

Gen, M. & Runwei Cheng. (1999). *Genetic Algorithms and Engineering Optimization | Wiley*. John Wiley & Sons. Retrieved from: https://www.wiley.com/en-us/Genetic+Algorithms+and+Engineering+Optimization-p-9780471315315.

Ghaznavi, M., Khan, A., Shahriar, N., Alsubhi, K., Ahmed, R. & Boutaba, R. (2015). Elastic virtual network function placement. *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*, pp. 255–260. doi: 10.1109/CloudNet.2015.7335318.

Ghribi, C., Mechtri, M. & Zeghlache, D. (2016). A Dynamic Programming Algorithm for Joint VNF Placement and Chaining. *Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking*, (CAN '16), 19–24. doi: 10.1145/3010079.3010083.

Gibbons, A. *Algorithmic Graph Theory | Algorithmics, complexity, computer algebra and computational geometry*. Cambridge university press. Retrieved from: https://www.cambridge.org/gb/academic/subjects/computer-science/algorithmics-complexity-computer-algebra-and-computational-g/algorithmic-graph-theory,https://www.cambridge.org/gb/academic/subjects/computer-science/algorithmics-complexity-computer-algebra-and-computational-g.

Gil Herrera, J. & Botero, J. F. (2016). Resource Allocation in NFV: A Comprehensive Survey. *IEEE Transactions on Network and Service Management*, 13(3), 518–532. doi: 10.1109/TNSM.2016.2598420.

Gouareb, R., Friderikos, V. & Aghvami, A. H. (2018). Delay Sensitive Virtual Network Function Placement and Routing. *2018 25th International Conference on Telecommunications (ICT)*, pp. 394–398. doi: 10.1109/ICT.2018.8464883.

Gurobi. Gurobi - The fastest solver. Retrieved from: https://www.gurobi.com/.

Hejja, K. & Hesselbach, X. (2019). Evaluating impacts of traffic migration and virtual network functions consolidation on power aware resource allocation algorithms. *Future Generation Computer Systems*, 101, 83–98. doi: 10.1016/j.future.2019.06.015.

Holland, J. H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT Press. Retrieved from: https://ieeexplore.ieee.org/book/6267401.

Houidi, O., Soualah, O., Louati, W., Mechtri, M., Zeghlache, D. & Kamoun, F. (2017). An Efficient Algorithm for Virtual Network Function Scaling. *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pp. 1–7. doi: 10.1109/GLOCOM.2017.8254727.

IBM. CPLEX Optimizer | IBM. Retrieved from: https://www.ibm.com/analytics/cplex-optimizer.

Ibn-Khedher, H., Abd-Elrahman, E., Afifi, H. & Forestier, J. (2015). Network issues in virtual machine migration. *2015 International Symposium on Networks, Computers and Communications (ISNCC)*, pp. 1–6. doi: 10.1109/ISNCC.2015.7238579.

Ibn-Khedher, H., Hadji, M., Abd-Elrahman, E., Afifi, H. & Kamal, A. E. (2016). Scalable and Cost Efficient Algorithms for Virtual CDN Migration. *2016 IEEE 41st Conference on Local Computer Networks (LCN)*, pp. 112–120. doi: 10.1109/LCN.2016.23.

Irving, R. W. (1985). An efficient algorithm for the "stable roommates" problem. *Journal of Algorithms*, 6(4), 577–595. doi: 10.1016/0196-6774(85)90033-1.

Jackson, K., Bunch, C. & Sigler, E. (2015). *OpenStack Cloud Computing Cookbook*. Packt Publishing Ltd.

Jami, V. & Reddy, G. R. M. (2016). A hybrid community detection based on evolutionary algorithms in social networks. *2016 IEEE Students' Conference on Electrical, Electronics and Computer Science (SCEECS)*, pp. 1–6. doi: 10.1109/SCEECS.2016.7509309.

Jungnickel, D. (2013). *Graphs, Networks and Algorithms* (ed. 4). Berlin Heidelberg: Springer-Verlag. doi: 10.1007/978-3-642-32278-5.

Juniper Networks. vMX Virtual Router Datasheet - Juniper Network. Retrieved from: https://www.juniper.net/us/en/products-services/routing/mx-series/datasheets/1000522.page.

Khai, N. T., Baumgartner, A. & Bauschert, T. (2019). Optimising Virtual Network Functions Migrations: A Flexible Multi-Step Approach. *2019 IEEE Conference on Network Softwarization (NetSoft)*, pp. 188–192. doi: 10.1109/NETSOFT.2019.8806710.

Khebbache, S., Hadji, M. & Zeghlache, D. (2017). Scalable and cost-efficient algorithms for VNF chaining and placement problem. *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, pp. 92–99. doi: 10.1109/ICIN.2017.7899395.

Khebbache, S., Hadji, M. & Zeghlache, D. (2018). A multi-objective non-dominated sorting genetic algorithm for VNF chains placement. *2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pp. 1–4. doi: 10.1109/CCNC.2018.8319250.

Kiji, N., Sato, T., Shinkuma, R. & Oki, E. (2020). Virtual network function placement and routing for multicast service chaining using merged paths. *Optical Switching and Networking*, 36, 100554. doi: 10.1016/j.osn.2020.100554.

Kubernetes. Production-Grade Container Orchestration. Retrieved from: https://kubernetes.io/.

Kutiel, G. & Rawitz, D. (2019). Service chain placement in SDNs. *Discrete Applied Mathematics*, 270, 168–180. doi: 10.1016/j.dam.2019.06.013.

Kwon, Changhyun. (2016). *Julia Programming for Operations Research*. Retrieved from: https://www.softcover.io/books/7b8eb7d0/juliabook.

Laaziz, L., Kara, N., Rabipour, R., Edstrom, C. & Lemieux, Y. (2019). FASTSCALE: A fast and scalable evolutionary algorithm for the joint placement and chaining of virtualized services. *Journal of Network and Computer Applications*, 148, 102429. doi: 10.1016/j.jnca.2019.102429.

Lam, A. Y. S. & Li, V. O. K. (2010). Chemical-Reaction-Inspired Metaheuristic for Optimization. *IEEE Transactions on Evolutionary Computation*, 14(3), 381–399. doi: 10.1109/TEVC.2009.2033580.

Lam, A. Y. S. & Li, V. O. K. (2012). Chemical Reaction Optimization: a tutorial. *Memetic Computing*, 4(1), 3–17. doi: 10.1007/s12293-012-0075-1.

Lee, Y. C. & Zomaya, A. Y. (2012). Energy efficient utilization of resources in cloud computing systems. *The Journal of Supercomputing*, 60(2), 268–280. doi: 10.1007/s11227-010-0421-3.

Liu, J., Li, Y., Zhang, Y., Su, L. & Jin, D. (2017). Improve Service Chaining Performance with Optimized Middlebox Placement. *IEEE Transactions on Services Computing*, 10(4), 560–573. doi: 10.1109/TSC.2015.2502252.

Liu, S. & Li, Z. (2017). A modified genetic algorithm for community detection in complex networks. *2017 International Conference on Algorithms, Methodology, Models and Applications in Emerging Technologies (ICAMMAET)*, pp. 1–3. doi: 10.1109/ICAMMAET.2017.8186747.

Luizelli, M. C., Bays, L. R., Buriol, L. S., Barcellos, M. P. & Gaspary, L. P. (2015a). Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions. *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 98–106. doi: 10.1109/INM.2015.7140281.

Luizelli, M. C., Bays, L. R., Buriol, L. S., Barcellos, M. P. & Gaspary, L. P. (2015b). Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions. *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 98–106. doi: 10.1109/INM.2015.7140281.

Luizelli, M. C., da Costa Cordeiro, W. L., Buriol, L. S. & Gaspary, L. P. (2017). A fix-and-optimize approach for efficient and large scale virtual network function placement and chaining. *Computer Communications*, 102, 67–77. doi: 10.1016/j.comcom.2016.11.002.

Ma, J., Rankothge, W., Makaya, C., Morales, M., Le, F. & Lobo, J. (2018). An Overview of A Load Balancer Architecture for VNF chains Horizontal Scaling. *2018 14th International Conference on Network and Service Management (CNSM)*, pp. 323–327.

Ma, S., Liu, S. & Meng, X. (2020). Optimized BP neural network algorithm for predicting ship trajectory. *2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, 1, 525–532. doi: 10.1109/ITNEC48623.2020.9085154.

Mechtri, M., Ghribi, C. & Zeghlache, D. (2016). A Scalable Algorithm for the Placement of Service Function Chains. *IEEE Transactions on Network and Service Management*, 13(3), 533–546. doi: 10.1109/TNSM.2016.2598068.

Medhi, M. P. a. D. (2004). *Routing, Flow, and Capacity Design in Communication and Computer Networks*. Elsevier. doi: 10.1016/B978-0-12-557189-0.X5000-8.

Mehmood, A., Muhammad, A., Ahmed Khan, T., Diaz Rivera, J. J., Iqbal, J., Ul Islam, I. & Song, W.-C. (2020). Energy-efficient auto-scaling of virtualized network function instances based on resource execution pattern. *Computers & Electrical Engineering*, 88, 106814. doi: 10.1016/j.compeleceng.2020.106814.

Mijumbi, R., Serrat, J., Gorricho, J., Bouten, N., Turck, F. D. & Boutaba, R. (2016). Network Function Virtualization: State-of-the-Art and Research Challenges. *IEEE Communications Surveys Tutorials*, 18(1), 236–262. doi: 10.1109/COMST.2015.2477041.

Miotto, G. (2017). NFVPEAER. Retrieved from: https://github.com/nfvpear/nfvpear.

Miotto, G., Luizelli, M. C., Cordeiro, W. L. d. C. & Gaspary, L. P. (2019). Adaptive placement & chaining of virtual network functions with NFV-PEAR. *Journal of Internet Services and Applications*, 10(1), 3. doi: 10.1186/s13174-019-0102-2.

Miyazawa, T., Jibiki, M., Kafle, V. P. & Harai, H. (2018). Autonomic resource arbitration and service-continuable network function migration along service function chains. *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–9. doi: 10.1109/NOMS.2018.8406235.

Muhammad, A., Sorkhoh, I., Qu, L. & Assi, C. (2021). Delay-Sensitive Multi-Source Multicast Resource Optimization in NFV-Enabled Networks: A Column Generation Approach. *IEEE Transactions on Network and Service Management*, 1–1. doi: 10.1109/TNSM.2021.3049718.

NetworkX. NetworkX. Retrieved from: https://networkx.org/.

Nguyen, T.-M., Minoux, M. & Fdida, S. (2019). Optimizing resource utilization in NFV dynamic systems: New exact and heuristic approaches. *Computer Networks*, 148, 129–141.

doi: 10.1016/j.comnet.2018.11.009.

Noghani, K. A., Kassler, A. & Taheri, J. (2019). On the Cost-Optimality Trade-off for Service Function Chain Reconfiguration. *2019 IEEE 8th International Conference on Cloud Networking (CloudNet)*, pp. 1–6. doi: 10.1109/CloudNet47604.2019.9064107.

OpenStack. OpenStack. Retrieved from: https://www.openstack.org/.

Padhy, S. & Chou, J. (2020). Finding the Optimal Reconfiguration for Network Function Virtualization Orchestration with Time-varied Workload. *Proceedings of the 3rd International Workshop on Systems and Network Telemetry and Analytics*, (SNTA '20), 49–52. doi: 10.1145/3391812.3396275.

Pei, j., Hong, P., Xue, K. & Li, D. (2018). Resource Aware Routing for Service Function Chains in SDN and NFV-Enabled Network. *IEEE Transactions on Services Computing*, 1–1. doi: 10.1109/TSC.2018.2849712.

Pei, J., Hong, P., Xue, K. & Li, D. (2019). Efficiently Embedding Service Function Chains with Dynamic Virtual Network Function Placement in Geo-Distributed Cloud System. *IEEE Transactions on Parallel and Distributed Systems*, 30(10), 2179–2192. doi: 10.1109/TPDS.2018.2880992.

Pham, C., Tran, N. H., Ren, S., Saad, W. & Hong, C. S. (2020). Traffic-Aware and Energy-Efficient vNF Placement for Service Chaining: Joint Sampling and Matching Approach. *IEEE Transactions on Services Computing*, 13(1), 172–185. doi: 10.1109/TSC.2017.2671867.

Prometheus. Prometheus - Monitoring system & time series database. Retrieved from: https://prometheus.io/.

Qazi, Z. A., Tu, C.-C., Chiang, L., Miao, R., Sekar, V. & Yu, M. (2013). SIMPLE-fying middlebox policy enforcement using SDN. *ACM SIGCOMM Computer Communication Review*, 43(4), 27–38. doi: 10.1145/2534169.2486022.

Qi, D., Shen, S. & Wang, G. (2019). Towards an efficient VNF placement in network function virtualization. *Computer Communications*, 138, 81–89. doi: 10.1016/j.comcom.2019.03.005.

Rankothge, W., Le, F., Russo, A. & Lobo, J. (2017). Optimizing Resource Allocation for Virtualized Network Functions in a Cloud Center Using Genetic Algorithms. *IEEE Transactions on Network and Service Management*, 14(2), 343–356. doi: 10.1109/TNSM.2017.2686979.

Rankothge, W., Ramalhinho, H. & Lobo, J. (2019). On the Scaling of Virtualized Network Functions. *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 125–133.

Sahni, S. & Gonzalez, T. (1976). P-Complete Approximation Problems. *Journal of the ACM*, 23(3), 555–565. doi: 10.1145/321958.321975.

Sang, Y., Ji, B., Gupta, G. R., Du, X. & Ye, L. (2017). Provably efficient algorithms for joint placement and allocation of virtual network functions. *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pp. 1–9. doi: 10.1109/INFOCOM.2017.8057036.

Sasabe, M. & Hara, T. (2020). Capacitated Shortest Path Tour Problem Based Integer Linear Programming for Service Chaining and Function Placement in NFV Networks. *IEEE Transactions on Network and Service Management*, 1–1. doi: 10.1109/TNSM.2020.3044329.

Sefraoui, O., Aissaoui, M. & Eleuldj, M. (2012). OpenStack: Toward an Open-source Solution for Cloud Computing. doi: 10.5120/8738-2991.

Sherry, J., Hasan, S., Scott, C., Krishnamurthy, A., Ratnasamy, S. & Sekar, V. (2012). Making middleboxes someone else's problem: network processing as a cloud service. *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, (SIGCOMM '12), 13–24. doi: 10.1145/2342356.2342359.

Son, J. & Buyya, R. (2019). Latency-aware Virtualized Network Function provisioning for distributed edge clouds. *Journal of Systems and Software*, 152, 24–31. doi: 10.1016/j.jss.2019.02.030.

Soualah, O., Mechtri, M., Ghribi, C. & Zeghlache, D. (2019). Online and batch algorithms for VNFs placement and chaining. *Computer Networks*, 158, 98–113. doi: 10.1016/j.comnet.2019.01.041.

Sugisono, K., Fukuoka, A. & Yamazaki, H. (2018). Migration for VNF Instances Forming Service Chain. *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*, pp. 1–3. doi: 10.1109/CloudNet.2018.8549194.

Sun, G., Li, Y., Yu, H., Vasilakos, A. V., Du, X. & Guizani, M. (2019). Energy-efficient and traffic-aware service function chaining orchestration in multi-domain networks. *Future Generation Computer Systems*, 91, 347–360. doi: 10.1016/j.future.2018.09.037.

Tashtarian, F., Zhani, M. F., Fatemipour, B. & Yazdani, D. (2020). CoDeC: A Cost-Effective and Delay-Aware SFC Deployment. *IEEE Transactions on Network and Service Management*, 17(2), 793–806. doi: 10.1109/TNSM.2019.2949753.

Torre, R., Urbano, E., Salah, H., Nguyen, G. T. & Fitzek, F. H. P. (2019). Towards a Better Understanding of Live Migration Performance with Docker Containers. *European Wireless 2019; 25th European Wireless Conference*, pp. 1–6.

Trajkovska, I., Kourtis, M.-A., Sakkas, C., Baudinot, D., Silva, J., Harsh, P., Xylouris, G., Bohnert, T. M. & Koumaras, H. (2017). SDN-based service function chaining mechanism and service prototype implementation in NFV scenario. *Computer Standards & Interfaces*, 54, 247–265. doi: 10.1016/j.csi.2017.01.002.

U-chupala, P., Uthayopas, P., Ichikawa, K., Date, S. & Abe, H. (2013). An implementation of a multi-site virtual cluster cloud. *The 2013 10th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, pp. 155–159. doi: 10.1109/JCSSE.2013.6567337.

Voorsluys, W., Broberg, J., Venugopal, S. & Buyya, R. (2009). Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation. *Cloud Computing*, (Lecture Notes in Computer Science), 254–265. doi: 10.1007/978-3-642-10665-1_23.

Wahab, O. A., Kara, N., Edstrom, C. & Lemieux, Y. (2019). MAPLE: A Machine Learning Approach for Efficient Placement and Adjustment of Virtual Network Functions. *Journal of Network and Computer Applications*, 142, 37–50. doi: 10.1016/j.jnca.2019.06.003.

Wang, P., Lan, J., Zhang, X., Hu, Y. & Chen, S. (2015). Dynamic function composition for network service chain: Model and optimization. *Computer Networks*, 92, 408–418. doi: 10.1016/j.comnet.2015.07.020.

Weerasiri, D., Barukh, M. C., Benatallah, B., Sheng, Q. Z. & Ranjan, R. (2017). A Taxonomy and Survey of Cloud Resource Orchestration Techniques. *ACM Computing Surveys*, 50(2), 26:1–26:41. doi: 10.1145/3054177.

Wu, Y., Su, Y. & Wen, C. H. (2017). TVM: Tabular VM migration for reducing hop violations of service chains in cloud datacenters. *2017 IEEE International Conference on Communications (ICC)*, pp. 1–6. doi: 10.1109/ICC.2017.7996680.

Xia, J., Cai, Z. & Xu, M. (2016). Optimized Virtual Network Functions Migration for NFV. *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 340–346. doi: 10.1109/ICPADS.2016.0053.

Xie, S., Ma, J. & Zhao, J. (2020a). FlexChain: Bridging Parallelism and Placement for Service Function Chains. *IEEE Transactions on Network and Service Management*, 1–1. doi: 10.1109/TNSM.2020.3047834.

Xie, Y., Wang, S. & Dai, Y. (2019). Provable Algorithm for Virtualised Network Function Chain Placement in Dynamic Environment. *2019 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6. doi: 10.1109/GLOBECOM38437.2019.9013582.

Xie, Y., Wang, S. & Dai, Y. (2020b). Revenue-maximizing virtualized network function chain placement in dynamic environment. *Future Generation Computer Systems*, 108, 650–661.

doi: 10.1016/j.future.2020.03.011.

Xing, H., Zhou, X., Wang, X., Luo, S., Dai, P., Li, K. & Yang, H. (2019). An integer encoding grey wolf optimizer for virtual network function placement. *Applied Soft Computing*, 76, 575–594. doi: 10.1016/j.asoc.2018.12.037.

Yang, S., Li, F., Trajanovski, S., Yahyapour, R. & Fu, X. (2021). Recent Advances of Resource Allocation in Network Function Virtualization. *IEEE Transactions on Parallel and Distributed Systems*, 32(2), 295–314. doi: 10.1109/TPDS.2020.3017001.

Ye, Q., Zhuang, W., Li, X. & Rao, J. (2019). End-to-End Delay Modeling for Embedded VNF Chains in 5G Core Networks. *IEEE Internet of Things Journal*, 6(1), 692–704. doi: 10.1109/JIOT.2018.2853708.

Yi, B., Wang, X., Li, K., Das, S. k. & Huang, M. (2018). A comprehensive survey of Network Function Virtualization. *Computer Networks*, 133, 212–262. doi: 10.1016/j.comnet.2018.01.021.

Yi, B., Wang, X., Huang, M. & Dong, A. (2019). A multi-criteria decision approach for minimizing the influence of VNF migration. *Computer Networks*, 159, 51–62. doi: 10.1016/j.comnet.2019.04.010.

Yu, H., Yang, J. & Fung, C. (2018). Elastic Network Service Chain with Fine-Grained Vertical Scaling. *2018 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–7. doi: 10.1109/GLOCOM.2018.8648096.

Zachary, W. W. (1977). An Information Flow Model for Conflict and Fission in Small Groups. *Journal of Anthropological Research*, 33(4), 452–473. doi: 10.1086/jar.33.4.3629752. Publisher: The University of Chicago Press.

Zhang, J., Li, L. & Wang, D. (2017). Optimizing VNF live migration via para-virtualization driver and QuickAssist technology. *2017 IEEE International Conference on Communications (ICC)*, pp. 1–6. doi: 10.1109/ICC.2017.7997166.

Zhou, X., Yi, B., Wang, X. & Huang, M. (2018). Approach for minimising network effect of VNF migration. *IET Communications*, 12(20), 2574–2581. doi: 10.1049/iet-com.2018.5188. Publisher: IET Digital Library.

Zhou, Y., Cheng, H. & Yu, J. X. (2009). Graph clustering based on structural/attribute similarities. *Proceedings of the VLDB Endowment*, 2(1), 718–729. doi: 10.14778/1687627.1687709.

Zou, J., Li, W., Wang, J., Qi, Q. & Sun, H. (2018). NFV Orchestration and Rapid Migration Based on Elastic Virtual Network and Container Technology. *2018 IEEE International*

Conference on Information Communication and Signal Processing (ICICSP), pp. 6–10. doi: 10.1109/ICICSP.2018.8549793.