Clustering Algorithms for Dynamic Adaptation of Service Function Chains

by

Imane EL MENSOUM

MANUSCRIPT-BASED THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE IN PARTIAL FULFILLMENT FOR A MASTER'S DEGREE WITH THESIS IN CONCENTRATION TELECOMMUNICATION NETWORKS
M.A.Sc

MONTRÉAL, DECEMBER 5, 2019

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
Département de génie logiciel et des TI, École de technologie supérieure

M. Aris Leivadeas, President of the Board of Examiners
Département de génie logiciel et des TI, École de technologie supérieure

M. Abdelouahed Gherbi, Member of the jury
Département de génie logiciel et des TI, École de technologie supérieure

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE PUBLIC

ON DECEMBER 2, 2019

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

Algorithmes de mise en grappes pour une adaptation dynamique des chaînes de services

Imane El MENSOUM

RÉSUMÉ

La virtualisation des fonctions réseau est l'un des paradigmes les plus adoptées dans les architectures réseau d'aujourd'hui, car elle offre entre autres une meilleure gestion des ressources, ainsi qu'une maintenance flexible des services déployés sur des ressources partagées dans des environnements en nuage.

Les fonctions réseau traditionnellement hébergées sur du matériel dédié sont désormais fournies comme des composantes logicielles, pouvant s'exécuter soit sur des machines virtuelles, soit sur des conteneurs. L'avantage majeur de cette transition c'est qu'elle facilite le déploiement des nouveaux services tout en optimisant le temps de gestion et d'administration des architectures réseau. Il est beaucoup plus facile de créer une nouvelle machine virtuelle/conteneur qui hébergera une fonction de réseau /service ou une application spécifique décrite sous forme de chaîne de services que de déployer un nouvel équipement matériel et de vérifier sa compatibilité avec le reste de l'architecture mise en place.

Avec tous les avantages qu'offre ce nouveau paradigme se présente un ensemble de défis principalement liés à : 1) l'optimisation de la consommation de ressources sur l'infrastructure partagée 2) la prise des meilleures décisions d'ordonnancement et de placement des fonctions virtuelles tout en respectant les exigences des clients ainsi que les ressources disponibles sur le réseau physique exprimées par différentes métriques (ex., CPU, mémoire, latence, bande passante, etc.)

Ce concept de la virtualisation des fonctions réseau (Network Function Virtualization – NFV), et plus spécifiquement du placement des chaines de services (Service Function Chains - SFC) a été traité dans de nombreux travaux de recherche proposant des approches garantissant un placement et un routage optimaux des fonctions de réseau virtualisées (Virtual Network Function – VNF) dans des réseaux physiques partagés, mais comme l'adoption de la virtualisation prend de plus en plus d'envergure et vu les exigences de plus en plus strict des applications d'aujourd'hui (ex., applications hautement sensibles à la latence, à la disponibilité de service, etc.), il est toujours important de considérer tous les paramètres ayant un impact sur la gestion du réseau dans les environnements Infonuagique.

Dans la cadre de ce projet de recherche, nous avons développé de nouvelles approches, pour le placement et le routage des fonctions réseau virtuelles dans des environnements Infonuagique. La première approche permet de former à la demande des grappes de serveurs déployés dans une l'infrastructure physique. Ces serveurs sont regroupés en fonction d'attributs similaires (ex., serveur à usage intensif en CPU, serveur efficace en énergie, etc.).

Ce processus constitue une mesure proactive permettant de s'assurer que les SFCs sont hébergées dans des serveurs garantissant leurs exigences en termes de métriques spécifiques (CPU, mémoire, disque…). Il utilise une méta-heuristique appelée CRO (Chemical Reaction Optimization) pour décider du meilleur placement des VNF garantissant une consommation de ressources optimale en termes de CPU / mémoire mais assurant également les latences les plus faibles lors du routage entre les différentes VNF, ce qui est un aspect important à considérer, car la plupart des applications actuelles demandent des latences faibles et des temps d'exécution les plus courts possible. Dans la deuxième approche, les grappes sont formées à l'aide d'algorithmes basés sur des méta-heuristiques, notamment le CRO, permettant d'améliorer la qualité des grappes constituées en termes de similarité, densité et modularité.

Mots-clés : virtualisation des fonctions réseau, chaines de services, méta-heuristique, allocation des ressources.

Clustering Algorithms for Dynamic Adaptation of Service Function Chains

Imane El MENSOUM

ABSTRACT

Network function virtualization is a pillar-stone of today's network architectures as it offers better management and elasticity and allows also a flexible maintenance of services running on shared resources over cloud environments.

Network functions traditionally hosted on dedicated hardware are now provided over software based components that might run either on virtual machines or on containers. The major advantage of this transition is that it makes the deployment of new services easier while optimizing the management and administration of network architectures. It is much easier to spin up a new virtual machine/container hosting a network function or a specific application described as a service function chain, than to deploy a new hardware based equipment and checking its compatibility with the rest of the architecture.

With all the advantages that this new paradigm offers comes a set of challenges related mainly to: 1) optimizing the resource consumption on the shared infrastructure 2) making the best decision of placing the virtual functions that respects at the same time clients' requirements and also leverages the available resources on the substrate network in terms of different metrics (e.g., CPU, memory, latency, bandwidth).

This aspect of Network Function Virtualization-NFV and Service Function Chains-SFC placement have been treated in so many research works that propose approaches ensuring optimal placement and chaining of VNFs in virtualized networks, but as the adoption of these technologies gets more important in real network setups, and given the strict restrictions of today's' applications (e.g. latency highly-sensitive applications, or availability highly-sensitive service, etc.), it is always important to consider all the parameters impacting the network management in cloud environments.

In this research project, we develop new approaches for placement and chaining of virtual network functions in cloud-based environments. The first approach allows forming on-demand clusters of servers deployed in a physical infrastructure. These servers are grouped according to their similar attributes (e.g., CPU-intensive server, energy-efficient server, etc).

This process is a proactive measure to ensure that SFCs are hosted in servers that meet their specific metrics requirements (CPU, memory, disk, etc.). It employs a meta-heuristic called CRO (Chemical Reaction Optimization) to decide of the best VNF placement guaranteeing optimal resource consumption in terms of CPU / memory. We employ CRO also to ensure the lowest latencies during the routing between the different VNFs. In fact, the E2E delay is an important aspect to consider, as most current applications require low latencies and shortest run times. In the second approach, the clusters are formed using algorithms based on

meta-heuristics, including the CRO, allowing to improve the quality of clusters formed in terms of similarity, density and modularity.

Keywords: Network function virtualization, Service function chains, metaheuristic, resource allocation.

TABLE OF CONTENT

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF ALGORITHMS

Page

LIST OF ABREVIATIONS AND ACRONYMS

CPU	Central processing unit
CRO	Chemical Reaction optimization
DC	Data Center
DS	Data Set
E2E	End to End delay
E-GT	Enhanced Game theory
GA	Genetic Algorithm
GT	Game Theory
ILP	Integer Linear Programming
MANO	Management and Orchestration
NFV	Network Function Virtualization
SDN	Software Defined Networks
SFC	Service Function Chain
SLA	Service Level Agreement
SLO	Service Level Objective
VNF	Virtual Network Function

LIST OF SYMBOLS AND UNITIES OF MEASURE

ms	millisecond
us	microsecond
s	second

INTRODUCTION

0.1 Context:

Today's networks are getting more complex due to the increasing demand of connectivity and speed rates by the consumers. The diversity of offered applications and services that are bandwidth intensive and that require important computation capabilities implies important evolutions in the network's architecture.

Networks today should be able to support multiple protocols, access technologies and different service layers. With all these complex needs and demands, it is getting harder for legacy networks to satisfy all these requirements along with ensuring an effective network management. Therefore, the adoption of virtualization techniques and software based networks becomes a must.

Network function virtualization (NFV) is an emerging concept that attracts increasing attention in the industry. NFV offers much more agility and flexibility to the network allowing easier software updates, resource adjustment and scalable changes in the configuration of the services. Network functions traditionally delivered on hardware and purpose-built platforms can now be provided through virtual resources which are called Virtual Network Functions (VNFs) and are hosted over shared physical network infrastructures.

Moreover, functions built on proprietary dedicated hardware are usually hard to upgrade, integrate and deploy over already built-up environments. This complexity is mainly due to compatibility issues coming from the use of different equipment manufacturers and vendors. Thus, using network functions as software components would help to improve the process of VNF deployment and the management cycle.

It would avoid having to check whether or not a function is compatible with the rest of the network's physical parts. In fact, VNFs are hardware independent which helps to reduce the maintenance costs that can increase drastically in case of major upgrades in the network infrastructure.

As defined by the European Telecommunications Standards Institute (ETSI) in Figure 0.1, the overall framework for NFV is composed of three main domains:

- The NFV infrastructure represents the combination of hardware and software resources which form the platform on which the VNFs would be deployed and run. The physical resources can be either storage nodes (servers) or network nodes like switches and routers providing the processing, storage and connectivity requirements to ensure the good functioning of virtual components.

- VNFs Virtual Network Functions: representing the software entities that implement the network functionalities over the NFV infrastructure. A network block might be instantiated over a single VM or multiple ones, a VNF can also have as many replicas as needed for resiliency constraints and to ensure the continuity of the service in case of hardware (servers) breakdown.

- NFV MANO Management and orchestration: This domain is responsible of the orchestration and management of the physical resources allocation as well as the virtual entities management during their lifecycles.

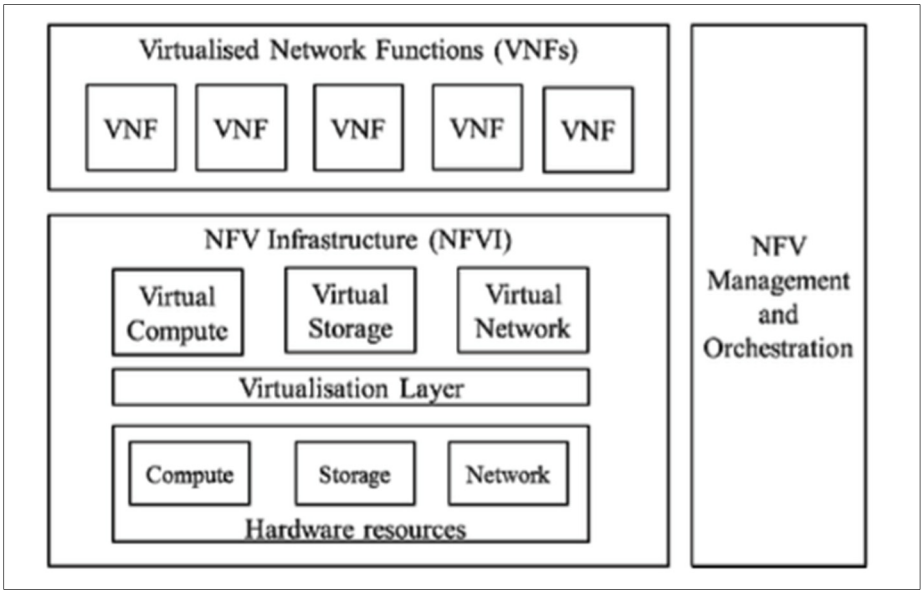


Figure 0.1 Network Functions Virtualization Framework
Taken from ETSI GS NFV 002 (2012, p.10)

With all the benefits that this framework offers, shifting to virtual based networks faces an important number of challenges. The main concerns are in terms of placing those virtual components, ensuring their interoperability and maintaining the quality of service at a good level. Hereby, we refer mainly to the NFV MANO orchestration and management, which is the targeted layer in this research project.

0.2 **Problem Definition:**

Combining Network Function Virtualization (NFV) and Software Defined Networks (SDN) concepts can bring many benefits in terms of automation, flexibility and agility in network management which justifies the keen interest they're getting in the IT sector. However, these new paradigms are a doubled-edged sword because they also come with serious challenges.

Shifting from legacy networks where every component of the network is a hardware entity to a virtual environment where functions are bricks of software programs is challenging. This transition requires a good management over the hosting servers so as to avoid wasting computation resources, storage capacity and energy consumption. The concrete deployment

of these virtual instances is a delicate task that is hindered by a set of technical issues some of them are related to: 1) how to efficiently place, readjust and schedule the VNFs of a certain network and 2) how to ensure that while doing this placement we make sure of the effective use of resources in the substrate network, whether in terms of storage and processing capacity, Service Level Objective (SLO) and/or energy consumption.

So, it comes back to finding a tradeoff between ensuring that all the VNFs instances are deployed with respect to the client requests and requirements. It is also important to optimize the use and the exploitation of the physical resources, so as to guarantee this promise of NFV/SDN paradigm to reduce the costs of network deployment and management.

These issues are considered as intriguing research questions and many contributions in the literature have been made to address them, which mainly focus on: 1) the placement of VNFs where the goal is to find the optimal set of servers to host all the incoming VNFs requests along with optimizing the consumption of hardware resources; 2) the service chains routing which is more related to the chaining between the set of VNFs composing a single SFC (both joint and non-joint placement and chaining might be considered). The chaining process guarantees low latencies, reduces the execution time to find the paths and minimizes the response time to clients' requests.

Even though many research works ((Mechtri et al.2016), (Pham et al.2018), (Khebbache et al.2018), (Draxler et al.2018), (Wang et al.2017)) have targeted this matter, there is still room for improvement and to overcome the limitations of existing solutions described in the literature.

Given the different parameters that may impact the quality of the placement and chaining decisions, it is usually hard to find a certain tradeoff between these several metrics (e.g. Energy, Memory, CPU, Delay, bandwidth, etc). The actual contributions reduce the complexity of this optimization problem by only selecting few metrics and do not take them all into account. Another aspect in the research works we have studied is that usually the

resource status is considered to be a static value, while in real case scenarios, the available resources in the network dynamically change, the same goes for the workload that evolves also continuously. The dynamic aspect of resource adaptation and traffic rates are important aspects that should also be considered while placing and chaining the virtual network functions.

To tackle these problematics, we defined two main sets of approaches: The 1st is related to clustering the substrate network into a set of on demand group of servers that are optimized in terms of energy, memory, or CPU. The clustering process is also dynamic and groups servers while optimizing similarity, modularity and density in these groups. This step is a pre-processing phase that helps not only to include several optimized metrics in the process of embedding SFCs, but also to reduce the search space for the 2nd set of approaches, which is related to placement and chaining. This 2nd track of our research is about the joint or non-joint placement and chaining of VNFs using evolutionary heuristics like the chemical reaction optimization and the genetic algorithms, these techniques will be further explained in the present report.

To the best of our knowledge none of the actual research works apply CRO-based joint (that optimizes both placement and chaining at the same time) or non-joint (that performs placement first then chaining after) algorithms on a set of pre-clustered substrate network to optimize resource consumption and reduce Service Level Agreement (SLA) violations as well. Moreover, no research work has been done actually to apply graph-based clustering technique in cloud-based environments and all the graph-based clustering papers we studied are more related to data analysis or community detection in social networks.

0.3 **Objectives:**

The main objectives of this research project are to develop two algorithms that might be integrated in the NFV management layer in order to classify the network in terms of physical

resources and optimally place and chain incoming SFC requests. The output of these algorithms should provide:

1. Homogenous groups of servers clustered together and have high similarities between them in terms of the selected attributes (e.g. Energy, Memory, CPU…) and with no outliers (servers that are not assigned to any cluster).

2. Dense clusters that are highly connected to ensure low latencies within the same group of servers and avoid violating the agreement in terms of delay/bandwidth.

3. Continuous adaptation of the clustering based on the variations of the resources on the servers and the workload state.

4. Optimal placement of the virtual network functions that ensure the respect of the clients' requests in terms of physical resources as well as avoiding resource wastage and over-provisioning in the physical infrastructure.

5. Optimal chaining of VNFs in the same SFC so as to respect the agreements in terms of delays and reduce the SLO violation.

0.4 **Methodology:**

As explained in the previous section, this project is divided into two major research tracks:

a. SFC Placement and chaining:

The first track concerns the definition of a heuristic approach for placement and chaining of virtual functions in cloud-based environments. The overall procedure for placement and chaining is is composed of three main steps. We start by classifying the network into groups of servers that are similar in terms of their attributes (which could be either energy, memory,

CPU, etc) using Integer Linear Programming (ILP) model adapted from the model proposed in (Abdul Wahab et al. (2019)). This proactive step ensures that the SFC requests will be placed on the servers responding to their specified conditions in terms of CPU, energy, etc. The clustering helps not only to satisfy the clients' requests, but also to reduce the search space for both phases of placement and chaining and thus reduce the overall execution time for our algorithm. Once this pre-processing phase is performed, we execute the CRO and GA heuristics to place the VNF components in a way that guarantees the SFC request but also optimizes the physical resource consumption in the provider's physical network. The last step is to ensure the chaining between the set of VNFs belonging to the same SFC, we used also CRO and GA heuristics for this step. The goal of the chaining is to ensure that the minimum end to end delay specified as an input by the client is respected. This is a non-joint placement and chaining approach. We also develop a CRO-based joint placement and chaining approach that is out of the scope of this thesis.

b. Clustering of the substrate network:

The second track is related to the dynamic clustering of the substrate networks. The aim of this work is to classify the servers in the substrate graph not only based on their attributes (e.g. CPU, Energy, memory...), but also based on the connectivity and the links between them. In this way, we ensure that not only servers in the same cluster are optimized in terms of the selected attributes, but their connectivity avoids violating the clients' requirements in terms of delay once the SFCs are deployed.

We developed a CRO-based approach to solve this graph-based clustering problem. We compared it with Game theory approach as well with optimal solution that considers the similarity between servers as well as the modularity which is the measure that ensures high density and connectivity between servers in the same cluster.

The clustering procedure is to be applied at the initialization phase of substrate network deployment but should be invoked at regular basis as the resources on the physical servers are continuously changing.

0.5 **Technical Contributions:**

The main motivations behind this work are to improve the management of resources in cloud-based environments while guaranteeing SLO as well as efficient and effective consumption of physical resources.

The project was divided in two major sets like we described in the Methodology section. The main contributions of this thesis are then:

1. Development of a multi-stage technique for SFCs placement and chaining: This was the subject of our first article entitled “MuSC: A Multi-Stage Service Chains Embedding Approach” accepted for publication in Journal of Network and Computer Applications, October 2019. In this joint research work, the main contribution of Miss Imane ElMensoum is the definition of ILP model adapted from the model defined in (Abdul Wahab et al. (2019)) and the development of non-joint CRO-based and GA-based placement and chaining approaches. The detailed description of this approach is presented in Chapter 2. A joint CRO-based and GA-based placement and chaining approaches have been also developed by Miss Imane El Mensoum. This contribution was submitted to IEEE Transactions on Services Computing. This work is out of the scope of this report.

2. Our second research contribution studies the graph-based clustering of substrate networks in order to host service functions chains. This second track was the subject of our second article entitled “VALKYRIE: A suite of topology-aware clustering approaches for cloud-based virtual network services” submitted to IEEE TRANSACTIONS ON CYBERNETICS Journal in October 2019. In addition, this

second approach was subject to a provisional patent application filled by the Patent Unit of Ericsson Canada to the United States Patent and Trademark Office (USPTO) in October 2019. In this joint research work, the contribution of Miss Imane El Mensoum is the development of CRO-based clustering technique that was compared to Game theory-based approach and optimal solution. You will find the article describing this approach in Chapter 3.

El Mensoum, I., Abdul Wahab, O., Kara, N., & Edstrom, C. (2019). MuSC: A Multi-Stage Service Chains Embedding Approach. Accepted for publication in Journal of Network and Computer Applications.

Lahlou, L., El Mensoum, I., Kara, N., & Edstrom, C. (2019). ARTIMIS: A chemical Reaction OptiMization approach for delay sensitive virtual network services. Submitted to IEEE Transactions on Services Computing, October 2019.

El Mensoum, I., Lahlou, L., A. Khasawneh, F., Kara, N., & Edstrom, C. (2019). VALKYRIE: A suite of topology-aware clustering approaches for cloud-based virtual network services Submitted to IEEE TRANSACTIONS ON CYBERNETICS Journal, November 2019.

El Mensoum, I., Lahlou, L., A. Khasawneh, F., Kara, N., & Edstrom, C. (2019). VALKYRIE: A suite of topology-aware clustering approaches for cloud-based virtual network services, Application Patent reference number P79104, October 2019

0.6 **Thesis Organization:**

The present thesis is structured by article, so we start first by a general literature review and then we include two distinct chapters to describe each of the pre-mentioned contributions.
In each of these chapters, we present a more detailed literature review related to the research tracks followed by the approach description, the used algorithms and the obtained results and their analysis. Chapter 4 is dedicated to an overall discussion of the results for both considered research matters, and last the conclusion and future work are discussed in the last section.

CHAPTER 1

LITERATURE REVIEW

Virtualization of network functions is a promising concept that attracts both cloud providers and also industry stakeholders, as it allows them to gain more flexibility to deploy new service function chains and also reduces the costs of instantiating and maintaining the overall quality of the offered service. With the arise of new technologies like 5G and IoT, it is of a great importance to ensure that this swap to virtualized infrastructures will be capable of respecting the requirements in terms of physical resources as well as SLO which are key performance indicators for every client's request.

Given all these challenges, many research works have targeted this field to tackle the resource management issue in cloud-based environments, and proposed approaches to deploy these SFCs with respect to the clients' requirements while at the same time optimizing the overall consumption of resources.

In this section, we will describe some of the previous research works that have treated the issue of service function chains embedding in cloud-based environments and analyze the proposed approaches aiming to optimize the resource consumption and improve the overall QoS of the offered service.

Most of the studies focus mainly on the optimal placement and chaining of virtual components of SFCs. The proposed works usually first apply the placement of VNFs followed by their chaining (non-joint placement and chaining).
This non-joint approach impacts the execution time of the full process, we propose in our approach to apply clustering as a pre-processing phase that helps us optimize the execution time as the substrate network is divided into a set of efficient clusters which makes the search space smaller for our metaheuristic-based algorithms designed for placement and chaining.

For instance, (Pham et al, 2018) proposes a non-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 (meaning physical resources 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 tradeoff. The approach presented in this article is based on a two-stage algorithm where the first step consists of finding the optimal set of servers to host the VNFs and then the second consists of placing them in a way to minimize the total cost, which means that the placement and chaining are performed in a sequential scheme and not in parallel.

In (Khebbache et al 2018), the authors propose a metaheuristic based on genetic algorithm NSGA-II to find Pareto optimal fronts that respect the considered metrics and find near optimal tradeoffs. As objective functions, the authors consider both the mapping cost in terms of resource utilization on hosting nodes as well as link resource consumption.

In (Khebbache et al 2017), the authors formulate the same problem in two approaches. The first one considers an exact mathematical model to solve the case of hosting SFCs composed of a maximum of 3 VNFs, the model considers resource consumption in terms of link utilization only. Obviously, this model cannot scale well with larger SFCs which is why the authors propose a heuristic-based approach to solve bigger instances and improve the runtime of the placement process of SFCs.

In (J.Pei et al 2018), the authors formulate the VNF chains placement problem as a Binary Integer Programming model (BIP) where the objective is to minimize the embedding cost that the authors define as the total of resource and placement costs. The resource cost represents the remaining resources on links, nodes and VNF instances while the placement cost represents the computing power, licenses fees and network utilization. As this problem is widely known to be NP-hard, the authors propose two algorithms to optimize the VNF placement. The first one is called SFC embedding Algorithm (SFC-MAP) used to place the

SFC requests with a minimum cost. The 2nd proposed algorithm is the VNF-DRA (VNF Dynamic Release Algorithm). As the network load has a dynamic nature it is important to consider this aspect so as to release redundant instances and reduce both resource consumption and running time. The VNF-DRA algorithm runs periodically and calculates the rate of resource usage on each VNF if it is lower than a defined threshold, then the considered VNF is either released or redirected, the threshold is computed based on the network load. Moreover, the authors include the lifetime of instances to drive a decision, for short lifetime SFC requests they're left to expire by their own and thus release the previously allocated resources to them. For long lifetime SFCs that are below the defined threshold, the algorithm first checks if they're not used at all they will be released and if they are lightly used their corresponding workload will be redirected to another instance of the same VNF.

Although, the problem of placement and chaining of SFCs has been widely addressed in the research area. The proposed approaches do not consider all the metrics that may impact the quality of the placement. Research works tend to reduce the complexity of the problem by defining objective functions only for specific metrics, and few of the studied works addressed the dynamic aspect of resources, while in real network scenarios the workload as well as the availability of resources are constantly changing and are also subject to peaks in some cases. That is why this aspect is to be taken considerably when developing new techniques for SFCs placement and chaining.

One way to reduce the complexity of the placement and chaining problem while at the same time considering several metrics for optimization is to include the clustering in the process. That is why we propose in our non-joint approach for placement and chaining of SFCs to use the clustering as it helps to cover a set of metrics that are treated prior to receiving new requests and reduce the runtime for the mapping of VNFs and their chaining.

Our approach is then capable of covering several metrics when performing the placement and chaining of VNFs while at the same time optimizing the run-time of this process which is an

important factor especially in cloud environments, where the resources are constantly changing, and decisions should be made as fast as possible.

To the best of our knowledge none of the actual proposed work has combined clustering techniques to form highly connected and similar groups of servers in terms of their attributes (energy, memory, CPU, etc,) with the placement and chaining procedures to cover more metrics in the optimization process and reduce the resource consumption on the substrate physical network. This is the reason why we were not able to compare our results to previous research works and we made the comparisons only between our adopted heuristics (Genetic Algorithm- GA, Chemical Reaction Optimization- CRO).

Moreover, most of the graph-based clustering techniques proposed in the literature are more related to data analysis or community detection in social networks. Therefore, and to the best of our knowledge, none of the actual research work define graph-based clustering technique for SFC placement and chaining in NFV environments.

The next two chapters are dedicated to each of the proposed contributions described in section 0.5. In each chapter, a more detailed literature review is described. The state-of-the-art section of Chapter 2 reviews research works related to the placement and chaining approaches while the literature review of Chapter 3 focuses on existing clustering approaches.

CHAPTER 2

MUSC: A MULTI-STAGE SERVICE CHAINS EMBEDDING APPROACH

Imane El Mensoum[a], Omar Abdel Wahab[b], Nadjia Kara[c], Claes Edstrom[d]

[a b c] Department of Software Engineering and IT, École de Technologie Supérieure, 1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[d]Ericsson Canada, 8275 Route Transcanadienne, Ville Saint-Laurent, Québec, Canada H4S 0B6

Paper accepted for publication in Elsevier Journal of Network and Computer Applications, October 2019

2.1 Abstract

Network function virtualization is an emerging concept that is attracting increasing attention in the industry because it offers high levels of agility and flexibility to the network allowing easier software updates, resource adjustment and scalable changes in the configuration of the services. Network functions traditionally delivered on hardware and purpose-built platforms in legacy networks can now be provided through shared virtual resources called VNFs (Virtual network functions) hosted over shared physical network infrastructures. Using network functions as absolute software components would certainly improve the deployment process and the management of the VNF life cycle. Since VNFs are hardware-independent, there is no need to check whether or not a network function is compatible with the rest of the network's physical parts; this also helps reduce the maintenance costs, which can increase drastically in the case of major upgrades to the network infrastructure. Along with all these benefits, shifting to virtual-based networks comes with a significant number of challenges, especially in terms of placing such virtual components, ensuring their interoperability and maintaining the quality of service at a level that is at least as good as what is offered by hardware based architectures. In this paper, we propose a cluster-based placement and chaining solution. The overall proposed approach consists of: 1) formulating an Integer Linear Programming (ILP) model aimed at finding an optimal tradeoff between multiple

objective functions that might be sometimes conflicting (e.g. hardware resource and energy consumption minimization, transmission delays, bandwidth usage, etc.), 2) classifying the substrate network into a set of on-demand clusters that are efficient for a predefined set of metrics, and 3) using meta-heuristic-based algorithms to find near-optimal solutions for the formulated ILP.

Keywords: Network Function Virtualization, Service function chains, Virtual functions placement and chaining, Genetic algorithm, Chemical Reaction Optimization.

2.2 Introduction

Network Function Virtualization (NFV) is based on the concept of moving network functions (e.g., routing, intrusion detection, etc.) from expensive physical hardware to software-oriented modules that are executed on top of commodity hardware in the form of Virtual Network Functions (VNFs). This approach has proven efficient in reducing both the Capital Expenditures (CapEx) and Operational Expenditures (OpEx) due to its resulting reduced equipment costs and reduced power consumption (Yi et al.(2018)). It also contributes in increasing the flexibility of scaling up, scaling down, and updating network services. However, the concrete deployment of NFV is hindered by a set of technical issues, relating mainly to the efficient placement and chaining of VNFs across physical nodes in the substrate network in order to minimize hardware resource consumption (Gil-Herrera et al.(2016)), while simultaneously respecting the dependencies among the VNFs composing a given network service. These objectives may be conflicting where, for example, the placement strategy that preserves the dependencies does not guarantee minimal hardware resource consumption or violates end-to-end delay requirements.

2.2.1 Problem Statement

There have been many efforts in the literature to address the VNF management issue, with contributions focusing mainly on two important aspects: the placement of the VNFs and their chaining. The first set of the proposed approaches (Xia et al.(2015), Mehraghdam et

al.(2014); Ghaznavi et al. (2015); Su et al.(2016)), seek to find the optimal set of servers on which to place each new incoming VNF, taking into consideration the optimization of several factors, such as hardware resource, and energy consumption. The second set (Alleg et al.(2017); Khebbache et al.(2018); Eramo et al. (2017)), for their part focus more on Service chains routing, and try to ensure that the chaining between the set of VNFs composing an SFC (assuming that the placement is already done) is realized in a way that guarantees low latencies, reduces run time to find the paths, and of course, minimizes the response time to clients' requests. One of the main limitations of these approaches stems from their static deployment scheme, which does not account for variations that are likely to occur at the level of server resources availability. These changes include sudden spikes or drops in the available resources on some servers, failure of some nodes, and changes in the SFC users' locations and needs. This leads to a degraded performance in dynamic settings wherein the status of the servers and the users' needs are continuously changing. For example, some servers that are the closest to some events (e.g. media servers used in festivals hosted by a city) in some time periods may experience unexpectedly high loads on their resources.

Therefore, incorporating dynamism into placement strategies is of prime importance to guarantee optimal Quality of Service (QoS) and resource consumption under changing environments. Another challenge faced in the proposed solutions is dealing with the complexity of the SFC placement and chaining problem; defining an optimal trade-off between multiple objective functions is usually hard to perform. Current research works reduce then the complexity of the problem, by defining their objective functions for few metrics (Energy/CPU consumption, latency, throughput, etc.) depending on the service chain priorities and its field of application. To address these challenges, our solution adopts a proactive approach in which the network is a priori divided into a set of efficient clusters to facilitate the placement and chaining processes in real-time. The clustering step helps not only to reduce the complexity of the problem, but also to cover a set of metrics, guaranteeing that the SFCs are placed based on their priorities among the predefined metrics, which could be either energy or memory, for instance. Once the clusters are formed, we employ our

heuristic-based solution to place and chain the requests in a dynamic scheme, taking into account variations in the network's available resources.

2.2.2 Contributions

We propose in this paper a cluster-based placement and chaining approach whose main premise is splitting the substrate network into a set of on-demand clusters that optimize some particular metrics (e.g., energy, memory, etc.). To perform the clustering, we employ the k-medoids clustering approach, while integrating statistical technique on top of it to improve the selection of the initial clusters. Thereafter, cluster-based placement and chaining algorithms are designed. Genetic Algorithm (GA) and Chemical Reaction Optimization (CRO) techniques are proposed to solve the placement and chaining problems, while considering several conflicting objectives, such as minimizing delays, CPU, energy, and memory consumption. This has the advantage of considerably reducing the solution space that CRO and GA meta-heuristics need to explore, thus accelerating both the placement and chaining processes.

The main contributions of this paper can be summarized in the following points:

- We formulate an Integer Linear Programming (ILP) to model the VNF placement and chaining problem, while taking into account the minimization of several metrics, such as hardware resource utilization, Service-Level Objective (SLO) violation cost and latency.
- We employ a modified k-medoids clustering approach to divide the substrate network into a set of on-demand clusters. The proposed clustering approach operates ina proactive manner to minimize the setup latency and decrease the complexity of placing/chaining VNFs.
- We develop cluster-based placement and chaining algorithms that employ meta-heuristic techniques such as GA and CRO to efficiently derive the appropriate VNF placement and chaining strategies. Two GA-based placement and chaining strategies are defined

and compared with CRO strategies: Classical-GA and Priority-based GA. These approaches are described in section 4.3.

- We perform experimental comparisons between the proposed algorithms to determine the advantages and limitations of each technique under different network scenarios.

Proceeding as such allows us to provide network administrators with a comprehensive view on what technique to use and under which conditions.

The rest of the paper is organized as follows: In the next section, we review research contributions that tackle the same problem of VNFs placement and chaining, and also highlight the particular features that distinguish our approach from the already proposed approaches in previous research works. In Section 3, we present our ILP model where we formulate the placement and chaining problem. Section 4 is where we discuss the different used algorithms and heuristics we use for clustering, placement and chaining. In Section 5, we present the simulations' setup, the results interpretation and discussion. Finally, in Section 6 we conclude our work and evaluate the performance of our proposed approach.

The rest of the paper is organized as follows: In the next section, we review research contributions that tackle the same problem of VNFs placement and chaining, and also highlight the particular features that distinguish our approach from the already proposed approaches in previous research works. In Section 3, we present our ILP model where we formulate the placement and chaining problem. Section 4 is where we discuss the different used algorithms and heuristics we use for clustering, placement and chaining. In Section 5, we present the simulations' setup, the results interpretation and discussion. Finally, in Section 6 we conclude our work and evaluate the performance of our proposed approach.

2.3 Related Work:

In this section, we carry out a detailed discussion of the major contributions that have been proposed concerning the VNF placement and chaining problem, and we highlight the unique

features of our solution in comparison with the state-of-the-art. Moreover, we provide an overview of some contributions in which CRO and GA have also been employed.

2.3.1 VNF Placement and Chaining approaches:

In (Mehraghdam et al. (2014)), the problem of optimally placing VNFs in the substrate network has been formulated as a Mixed Integer Quadratically Constrained Program (MIQCP) to derive the optimal placement and chaining of network functions, while considering the limited network resources and functional requirements. In (Cohen et al. (2015)), the authors formulate the VNF placement problem as an integer linear programming model with the aim of minimizing both the deployment and connection (between VNFs) costs and propose some approximation algorithms to solve it. In (Khebbache et al. (2018)), the authors formulate the VNF placing problem as a multi objective optimization problem aimed at minimizing two major metrics: the mapping cost and physical link utilization. A meta-heuristic based on the Non-Dominated Sorting Genetic Algorithm II (NSGA-II) is proposed by the authors to find near-optimal solutions respecting the considered metrics and ensuring acceptable performance trade-offs. In (Pham et al. (2017)), the authors propose a joint placement and chaining approach to solve the problem using sampling-based Markov approximation combined with matching theory. In order 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 in which the weights set for each cost function are defined by the provider and can be adjusted to achieve the desired tradeoff.

In (Ghaznavi et al. (2015)), the authors formulate an Elastic Virtual Network Function Placement (EVNFP) problem and derive a solution to minimize the operation cost, while considering the minimization of elasticity overhead and the trade-off between bandwidth and host resource consumption. Thereafter, a heuristic solution called Simple Lazy Facility Location (SLFL) is advanced to derive the appropriate placement of VNFs. The authors consider four challenges related to NFV, namely: (1) where to place Virtual Network Function Instances (VNFIs), (2) how many resources should be assigned to each VNFI, (3)

how to convey SFC requests to the adequate VNFIs in the right order, and (4) how to readjust VNFIs to cope with the sudden changes in SFC requests concentration. To tackle these challenges, they proposed a three-phase solution whose objectives are to (1) minimize the SFC bandwidth rejection, and (2) minimize the energy consumption by reducing the number of running servers. In (Alleg et al. (2017)), the authors proposed a delay-aware placement and chaining model formulated as a Mixed Integer Quadratically Constrained problem (MIQCP). The presented approach aims at ensuring a flexible allocation of resources in the substrate network. The main idea behind this work is to ensure that the proposed solutions are capable of answering the SFC requests in terms of end to end delay, without having to allocate unnecessary physical resources on the hosting nodes, causing an over-dimensioning of the virtual network and thus leading to a significant increase in operational costs for the service provider. In (Liu et al. (2017)), the authors address the challenge of jointly optimizing the placement of new incoming SFCs and readjusting in service ones. These cases are modeled as an integer linear programming model that aims to minimize both the operational overhead and readjustment cost. To solve this problem, a column generation approach is employed. In (Draxler et al. (2018)), the authors introduce new scheme to formulate the SFC request, unlike the common representation of an SFC request consisting of a directed weighted graph composed of a set of VNF linked to each other. In this work, the authors consider that each SFC is described by a template, containing information about the components and the interconnection between them only, while for resource requirements they are considered as a function of the incoming data rates on each component. This representation allows the adaptation of resources according to the flow rates variation. While this approach may be considered dynamic, it doesn't scale well with large networks. According to the simulations, the authors argue that it can be used for geographically distributed architectures where each node represents a whole data center. In (Lange et al. (2017)), a multi objective heuristics proposed to solve the VNF chains placement problem, the adopted approach is based on the simulated annealing algorithm. This technique aims to find feasible solutions that respect the set of objectives in terms of CPU use and link resource consumption. The proposed algorithm solves the VNF placement problem in two stages: assigning the requests to adequate nodes, and then routing of the demands. Most of the

contributions above focus on the optimization of VNFs placement and chaining on the substrate network, while considering different objective functions. Still, most of these works either ignore the dynamic aspect of resources in cloud-based environments or do not consider large substrate networks, but use relatively small architectures for their simulations. However, in order to benefit from the advantages offered by NFV, these challenges must be considered. In fact, the resource consumption and availability of the servers in datacenters are continuously changing depending on the adopted placement strategy, the number of served users and their locations. Moreover, VNFs should be continuously adjusted to guarantee service continuity as users are constantly changing their locations over large-scale network topologies.

To deal with these problems, our approach adopts a proactive process in which the network is a priori divided into a set of on-demand clusters optimized for certain defined metrics (e.g. energy, memory, etc.,), and that aims to facilitate the placement and chaining process. Network resource status may be monitored by an SDN controller to keep track of resource variations. This information is provided to our heuristic-based solutions as an input to ensure efficient initial placement and chaining, and may also be used for readjustment purposes.

2.3.2 CRO and GA Based Approaches:

In (Xu et al. (2011)), the authors tackle the problem of task scheduling in grid computing systems. To this end, they formulate an optimization problem aimed at minimizing the Make span and Flow time of the scheduling process, while accounting for resource reliability. To solve the problem efficiently, they employ several versions of CRO to derive effective scheduling solutions. In (Lam et al. (2010)), the authors take advantage of CRO to address the problem of allocating the available channels to unlicensed users in cognitive radio scenarios. They formulated an optimization problem with three utility functions that aims to minimize hardware utilization, while guaranteeing the fairness of the allocation process among users. Genetic algorithms have been extensively used in the context of NFV. For example, in (Dezhabad et al. (2018)), the authors discuss a method to achieve dynamic auto-

scalability of virtual firewalls. The objective is to determine the number of active virtual firewalls needed at each time period, based on the density of the incoming load and the portion of requests that should be assigned to each single firewall. The problem is solved using a hybrid method, which combines a genetic algorithm and reinforcement learning. The authors of (Yala et al. (2018)) propose a placement strategy for Multi-access Edge Computing (MEC) in NFV environments. They formulate the problem as an optimization problem with two conflicting objectives, i.e., maximizing service availability and minimizing access latency. A genetic algorithm is then advanced to solve the optimization problem. Although CRO and GA have been widely used to model a variety of solutions, this work is, to the best of our knowledge, the first that combines them in a cluster based environment to tackle the problem of placing and chaining SFCs.

2.4 Problem formulation:

We consider an undirected graph $G_s(V_h,E)$ to represent the substrate network, where V_h represents the set of substrate nodes, while E is the set of links connecting these nodes. The substrate nodes can be categorized in two different sets, namely, host servers and middleboxes. Host servers are high-volume servers used to host Virtual Machines (VMs) or containers that may belong to one or different users. Middleboxes are physical network components providing various network functions, like Wide Area Network (WAN) optimizers, multimedia caches or Intrusion Detection Systems (IDSs) and Intrusion Prevention Systems (IPSs). $G_v(V_v;E_v)$ is a directed acyclic graph that represents an SFC composed of a sequence V_v of VNFs and a set E_v of virtual links connecting the VNFs. In this section, we formulate our proposed ILP to model the VNF placement and chaining problem adapted from the model proposed in (Abdul Wahab et al. (2019)). Table 2.1 summarizes the different notations used throughout the paper.

Table 2.1 Notations

Symbol Significance	
V_h	Set of host servers in the substrate network.
V_v	Set of virtual network functions in a certain SFC.
E	Set of physical links in the substrate network.
E_v	Set of virtual links in a certain SFC.
R_{vh}^{max}	Amount of hardware resource capacity on physical server $v_{h\in} V_h$
R_m^p	Amount of hardware resources consumed by VNF of type $p_{\in} P$ on server m
$\alpha_{n,m}^p$	Decision variable whose value is equal to 1 if VNF n of type p is hosted in server m and 0 otherwise.
$\beta_{i,j}$	Decision variable whose value is 1 if the virtual link i is mapped into physical link j and 0 otherwise.
$\xi_{n,m}$	Monetary weight of adding a new VNF n of physical server m.
$\Upsilon_{x,y}^{a,b}$	Monetary weight of the service delay penalty between each pair of hosted VNFs a and b hosted on physical servers xand y respectively.
$d_{x,y}^{a,b}(t)$	The propagation delay between the pair (a, b) of VNFs hosted on physical servers x and y respectively.
$N^{\lvert Vv\rvert}$	Number of VNFs $\lvert V_v\rvert$ hosted on physical server m.
$l_{x,y}^{a,b}$	Maximum delay cost that a flow of pair a and b of VNFs hosted on topof physical servers x and y respectively can tolerate.
Π	Set of metrics to be minimized.
h^{θ}	Cluster-head h that is efficient in terms of metrics θ (where $\theta \subseteq \prod$ is a single combination subset of the set $\prod$).
$\phi_{Vvk\ (s)}$	Set of VNFs V_{vk} hosted on physical server
$\Psi_{Ev}(e)$	Set of virtual links E_v hosted on substrate link e

Cost Functions:

Operational Cost: The operational cost in our case represents the cost of adding new VNFs since the last service time. Specifically, the operational cost can be computed using Eq. (2.1)

$$C_{opr}(t) = \sum_{n=1}^{|Vv|} \sum_{m=1}^{|Vh|} \alpha_{a,x}^{p}(t).\xi_{n,m}(t).\left[N_m^{|Vv|}(t) - N_m^{|Vv|}(t-1)\right], \forall\, p \in P \tag{2.1}$$

This function aims at minimizing the resource consumption at the substrate level and thus reduces the monetary costs of adding new VNFs.

Penalty Cost: The cost of traffic transmission of an accepted SFC through a physical link in the substrate network is formulated in Eq.(2.2)

$$C_{penalty}(t) = \sum_{a=1}^{|Vv|} \sum_{x=1}^{|Vh|} \sum_{b=1}^{|Vv|} \sum_{y=1}^{|Vh|} \alpha_{a,x}^{p}(t).\alpha_{b,y}^{p}(t).\mathrm{d}_{x,y}^{a,b}(t).\Upsilon_{x,y}^{a,b}(t)\,, \forall\, p \in P \tag{2.2}$$

This function aims at minimizing the total delay between VNFs of the same SFC to avoid exceeding the limit set by the clients request.

Objective Functions:

The cost function C(t) of the provider at a certain time moment t can then be defined as the sum of the two previous costs: the operational and penalty costs at that time period, i.e.,

$$C(t) = C_{opr}(t) + C_{penalty}(t) \tag{2.3}$$

Thus, the objective of the providers is to:

$$minimize \sum_{t=t1}^{tn} C(t) \tag{2.4}$$

VNF Assignment and Chaining Constraints:

Constraint (2.5) ensures that the hardware resources consumed by the VNFs cannot exceed the physical server's hardware resources capacity.

$$\sum_{t=t1}^{tn} \sum_{n=1}^{|Vv|} \sum_{m=1}^{|Vh|} \alpha_{n,m}^{p}(t).\mathrm{R}_{m}^{p}(t) \leq \mathrm{R}_{m}^{max}(t)\ , \forall\ p\ \in P \tag{2.5}$$

Constraint (2.6) ensures that the total operational cost at each time moment in the time interval [t_1, t_n] does not go beyond an upper limit O^{max} defined by the provider in order to optimize its own profit.

$$\sum_{t=t1}^{tn} C_{opr}(t) \leq O^{max}(t) \tag{2.6}$$

Constraint (2.7) defines the penalty cost to guarantee that the delay of the flow of each pair of VNFs, cannot exceed the maximum limit that this flow can tolerate.

$$\sum_{t=t1}^{tn} C_{penalty}(t) \leq l_{x,y}^{a,b}(t), \forall\ a,b\ \in Vv\ and\ x,y\ \in Vh \tag{2.7}$$

Constraint (2.8) guarantees that each virtual link e_v in E_v is mapped to only one single link on the substrate network e in E.

$$\sum_{t=t1}^{tn} \sum_{e\in E} \sum_{ev\in Ev} \beta_{ev,e}(t) = 1 \tag{2.8}$$

Constraint (2.9) guarantees that each VNF instance n in V_v of type p is mapped to only one single server in the substrate network m in V_h.

$$\sum_{t=t1}^{tn} \sum_{m \in Vh} \alpha_{n,m}^{p}(t) = 1, \forall\, n \in Vv \tag{2.9}$$

2.5 Algorithms:

2.5.1 K-medoids Clustering process:

The clustering process consists of two principal steps: The first step enables selecting cluster heads, while the second allows distributing the substrate nodes to their corresponding clusters. The proposed method to optimize the selection of the initial set of cluster heads is a statistical technique. The main idea of this technique is to exclude the outliers (e.g., in terms of energy) that are too far from the central region, which have high variance values. Once the initial cluster heads are defined, the remaining servers are then assigned to the closest cluster-head (in terms of the considered attributes). After that, for each cluster, the server which minimizes the distance from the other servers in the same cluster is selected as a replacement cluster-head. Consequently, each server is assigned again to the closest newly appointed cluster-head and the new clustering cost is computed once again. If the new clustering cost is equal to the previous one, the algorithm stops and the clustering topology remains the same. Otherwise, the procedure is repeated until reaching a stable clustering topology. The distance we are referring to in this clustering phase can be calculated in terms of different metrics depending on the SFCs' requests and priorities. In our approach simulation, we will tackle clustering in terms of energy and memory, but these metrics can be easily adapted to include other substrate servers' parameters. The full procedure for clustering is summarized in appendix A.

2.5.2 Overall Cluster-based SFC Placement and Chaining:

The clustering phase reduces the complexity of our objective functions defined in Equation (4). The next step is then to execute VNF placement and chaining processes, while taking

advantage of the clustering results. Placement and chaining processes are distributed on each of the selected cluster heads formed according to Algorithm A-1, which reduces the search space (i.e., the number of servers and links) of the problem. More specifically, instead of having a single central module responsible for placing and chaining all of the VNFs on the total set of servers and links, each cluster head would only be executing these steps for a subset of the original substrate network. The cluster-based placement and chaining process is described in Algorithm 2.1. Each incoming SFC will be assigned to its appropriate cluster, and the corresponding cluster head will solve the ILP described in Section 3 using metaheuristic approaches described in sections 4.3 and 4.4 (using Algorithm 2.17 for placement and Algorithm 2.7 for chaining) - only for its cluster members -in order to determine the VNF-to-Server mappings. Based on the results obtained by the solution of the ILP, the VNFs composing the underlying SFC are placed on the servers chosen by the feasible solution (Algorithm 2.1 - Steps 6-11) and the virtual links among the VNFs are mapped as well to the corresponding physical links as well.

Algorithm 2.1 Placement and chaining algorithm

Input: Set $\prod$= set of metrics to be minimized (e.g., {energy, memory ,CPU, delay} .

Cluster-head h^{θ} that is efficient in terms of metrics θ(where$\theta \subseteq \prod$is a single combination subset of the set $\prod$) and responsible for executing the Algorithm.

Service function chain composed of a set $V_v^k = \{v_v^1, \ldots, v_v^n\}$of VNFs

Set $\phi_{V_v^k(Si)}$of VNFs V_v^k hosted on each server S_i

Set $\Psi_{Ev}(e)$of virtual links E_v hosted on substrate link e.

Output: Updated set$\phi'_{V_v^k(Si)}$ of VNFs V_v^k hosted on each server S_i

Updated set $\Psi'_{Ev}(e)$ of virtual links E_v mapped to link e

Procedure PLACEMENT & CHAINING

1. Run Algorithm 2.17 and 2.7 to solve the ILP in Eq. (2.4) for the metrics $\Pi = \Pi \backslash \Pi \cap \theta$
2. for each server $S_i \in V_h$
3. for each VNF $v_v \in V_v$
   if $\alpha^{p}_{v_v,Si} == 1$ then
   Assign v_v^i to server Si, i.e., $\phi_{V_v^k(Si)} = \phi_{V_v^k(Si)} \cup v_v^i$
   end if
4. **end for**
5. **end for**
6. **for** each link $e \in E$
7. **for** each virtual link $e_v \in E_v$
   If $\beta_{ev,v} == 1$then
   Assign e_v to link e ,i.e., $\Psi_{Ev}(e) = \Psi_{Ev}(e) \cup e_v$
   end if
8. **end for**
9. **end for**
10. $\boldsymbol{\phi'_{V_v^k(Si)} = \phi_{V_v^k(Si)}}$
11. $\Psi'_{Ev}(e) = \Psi_{Ev}(e)$
12. **end procedure**

2.5.3 Metaheuristic Chaining approaches:

2.5.3.1 Chemical Reaction Optimization approach:

CRO is a nature-inspired approach that tries to map chemical reactions to optimization problems. The base unit in CRO is the molecule. Each molecule represents a possible solution to the underlying optimization problem and enjoys several attributes such as the molecular structure w, the potential energy PE, the Kinetic energy (KE), and the number of hits (Num-Hits). Next, we explain each of these attributes in detail:

Molecular structure w: represents a solution (i.e., number, vector, or matrix) of the underlying problem.

Potential energy (PE): represents the value of the objective function of the solution represented by w, i.e., $PE_w = f(w)$, with f being the objective function.

Kinetic energy (KE): quantifies the indulgence of the system in accepting a solution that is worse than the existing one.

Number of hits (Num-Hits): is a counter of the total number of hits (also known as collisions) that a certain molecule has undergone so far.

A chemical system, composed of a set of molecules, experiences a chemical reaction if it happens to hold excessive energy. In other words, the chemical system releases the excessive energy in order to stabilize itself. This change is triggered by a collision. A collision can be of one of two types: uni-molecular and intermolecular. Uni-molecular collisions occur when a molecule hits an external material (e.g., a wall of the container). Inter-molecular collisions, on the other hand, occur when molecules collide with each other. Every collision results in a corresponding reaction change referred to as an elementary reaction. Four types of elementary reactions can be exploited to manipulate solutions (i.e., explore the solution space):

On-Wall Ineffective Collision: This type of reactions occurs when a certain molecule hits a wall of the container and then bounces away resulting in one single unit. In this type of collisions, the existing molecular structure w is perturbed and becomes w'.

Decomposition: This type of reactions occurs when a certain molecule hits a wall, resulting in the molecule breaking into several parts, i.e., $w = w_1 + w_2$

Inter-Molecular Ineffective Collision: This type of reactions occurs when several molecules collide with one another, and then bounce away. In such collisions, the structure of the involved molecules (say w1 and w2) is perturbed and becomes w'_1 and w'_2 , i.e.,
$w_1 + w_2 = w'_1 + w'_2$

Synthesis: The effect of this type of reactions is the opposite of that of the decomposition phase. Specifically, the synthesis occurs when several molecules collide with each other and agglomerate into one molecule, i.e., $w_1 + w_2 = w'$

Next, we explain how we employed the CRO to solve our ILP described in Section 3. The molecule encoding, we used for our CRO-based algorithm is illustrated in Figure 2.1, with the path represented by an array containing the set of nodes representing the flow path to go from the source node S to the destination node D (where our VNFs are hosted).

The Algorithm 2.2 allows generating a set of initial solutions for the delay minimization problem. This is done by starting the construction of the path from the intended source node and then continuously picking a random node from the neighborhood, provided that this node is different from the destination node, and was not previously selected in the path. This process is repeated continuously while changing the index of the source node after each addition of a new member until reaching the destination node.

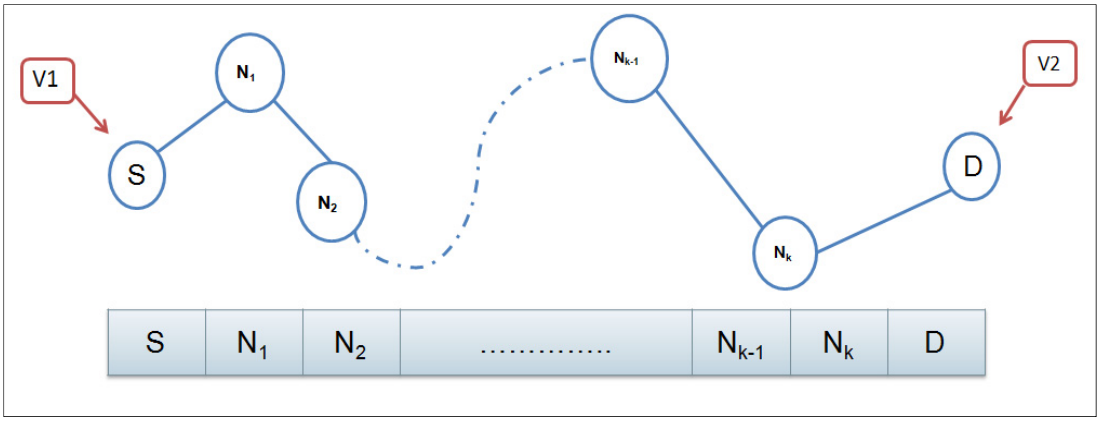


Figure 2.1 CRO Molecule Encoding

Algorithm 2.2 Molecules Generation

Input: Set S_i of all the nodes that are adjacent to the node i

Source node Src

Destination node Dst

Output: Path vector P

Procedure GENERATE MOLECULES

1. Initialize index i = 0
2. Construct the path P starting from the source node, i.e., P[i] = Src
3. **While** P[i]≠Dst do

   Randomly pick a neighbor node n from

   If n ∈ P

   Randomly pick another neighbor such that n'∈ $S_{P[i]}$ and n'≠ n

   else

   increment i, i.e., i++

   P[i] = n

   end if
4. **end while**
5. **Return** P
6. **end procedure**

In Algorithm 2.3, we describe the synthesis reaction. The inputs to this algorithm are two molecules P_1 and P_2 representing two possible solutions, and the output is another molecule P_s representing a possible solution after some manipulations. In the first step of the algorithm, the common nodes between the two input molecules are retrieved and saved into C (Steps 1-4 - Algorithm 2.3). Thereafter, a new molecule is constructed by bringing together parts from both P_1 and P_2 i.e., $Ps = P_1[1 : C_1] \cup P_2[C2 +1 : length(P_2)]$. Finally, the delay entailed by the newly formed molecule P_s is compared with that of the initial molecules P_1 and P_2 (Step 8 -Algorithm 2.3). If the delay of P_s is lower than that of the two initial solutions P_1 and P_2, then these two latter molecules are destroyed and only P_s is kept as the output of the Algorithm; otherwise, P_s is destroyed and P_1 and P_2 are kept in the population.

Algorithm 2.3 Synthesis

Input: Two molecules P_1 and P_2

procedure Synthesis

1. **for** i: length (P_1)
2. **for** j: length(P_2)
   if P_1[i] is equal to P_2[j]
   Construct potential intersection nodes C = {i,j}
   end if
3. **end for**
4. **end for**
5. Randomly choose a couple of nodes {C_1 ,C_2 } from C
6. Construct the set P_s such that P_s=P_1[1: Ci] UP_2[C_2+1: length(P_2)]
7. Compute thetotal delay of P_s
8. **If** delay (P_s)< delay (P_1)|| delay (P_s)< delay (P_2)
9. Destroy the molecules P_1 and P_2
10. Add P_s to the population
11. **else**
12. Destroy the molecule P_s
13. **end if**
14. **end procedure**

The inter-molecular ineffective collision process is illustrated in Algorithm 2.4. The inputs to the Algorithm are two molecules P_1 and P_2 representing two possible solutions and the outputs are two other molecules representing two other possible solutions after some manipulations. In the first step, a node N_{k1} is randomly selected form the molecule P_1 (Step 1 - Algorithm 2.4). A new path P' is then constructed using Algorithm 2.2 starting from the randomly chosen node (Step 2 - Algorithm 2.4), and new molecule P_3 is generated such that P3 = P1[1 : Nk1] U P' (Step 3 - Algorithm 2.4).

Similarly, another node N_{k2} is randomly selected form the molecule P_2 (Step 5 - Algorithm 2.4), another new path P'' is constructed using Algorithm 2 starting from N_{k2} (Steps 6 - Algorithm 2.4), and a new molecule P_4 is generated such that P4 = P2[1: Nk2]U P'' (Step 7 - Algorithm 2.4). The delay of the newly generated molecules is then compared against that of the initial molecules (Step 9 -Algorithm 2.4). If the delay of the newly generated molecules is lower than that of the initial ones, then those initial molecules are destroyed and the new ones are returned by the algorithm (Step10 - Algorithm 2.4). Otherwise, the new molecules are destroyed, and the initial ones are returned by the algorithm (Step 13 – Algorithm 2.4).

Algorithm 2.4 Inter-Molecular Ineffective Collision

Input: Two molecules P_1 and P_2
Procedure Inter-Molecular Ineffective Collision
1. Randomly select a node from the molecule A
2. Generate a path P' starting from node to destination node using Algorithm 2
3. Construct the molecule $P_3 = P_1[1: N_k^1] \cup P'$
4. Calculate total delay for P_3
5. Randomly select one node N_k^2 from P_2
6. Generate a path P'' starting from node $N_k^2$to destination node usingAlgorithm 2
7. Construct the set $P_4 = P_2[1: N_k^2] \cup P''$
8. Calculate total delay for P_4
9. **if** delay (P_3)< delay (P_1) *and* delay (P_4)< delay (P_2)
10. Destroy the molecules P_1 and P_2
11. Add the molecules P_3 and P_4 to the population
12. **else**
13. Destroy the molecules $P3$ and $P4$
14. **end if**

In Algorithm 2.5, the on-wall ineffective collision process for the delay maximization problem is described. The input to this Algorithm is a molecule P_1 representing a possible solution to the problem and the output is another molecule P_2 which represents the possible solution after some perturbations. In the first step of the algorithm, a node N_k from the input molecule is randomly chosen (Step 1 - Algorithm 2.5). Then, a new molecule is partially constructed such that $P_2 = P_1[1 : N_k]$. The rest of the molecule is then populated using Algorithm 2.2. If the delay of the newly created molecule is lower than that of the initial molecule, then the initial molecule is destroyed and the newly created molecule is returned by the algorithm (Steps 4-6 - Algorithm 2.5). Otherwise, the new molecule is destroyed and the initial molecule is kept as the output of the algorithm (Step 8 - Algorithm 2.5).

Algorithm 2.5 On-Wall Ineffective Collision

Input: Molecule P_1

procedure On-Wall Ineffective Collision

1. Randomly select a node Nk from the molecule P_1
2. Construct the molecule $P_2 = P_1$ [1: Nk] and calculate the rest of the path using Algorithm 2
3. Calculate total delay of P_2
4. **if** delay(P_2) < delay(P_1)
5. Destroy P_1
6. Add P2 to the population
7. **else**
8. Destroy P_2
9. **end if**

The decomposition process for the delay maximization problem is described in Algorithm 2.6. The inputs to this Algorithm are a molecule P_1 representing a possible solution to the problem and its length L, and the outputs are two molecules P_2 and P_3 generated through the decomposition of P_1. First, the algorithm randomly selects a node N_k from the input molecule

(Step 1 -Algorithm 2.6). A new path P' is then constructed using Algorithm 2.2 starting from the randomly chosen node N_k to the destination node (Step 2 - Algorithm 2.6). Thereafter, a new molecule P_2 is constructed such that $P_2 = P_1[1 : N_k]U\ P'$ (Step 3 - Algorithm 2.6). Similarly, a new molecule P_3 is constructed using Algorithm 2 starting from the source to node N_k and then a new molecule P_3 is constructed where $P_3 = P'' \cup P_1[N_k + 1 : L]$ (Step 5 – Algorithm 2.6) The delays entailed by the newly generated molecules are then compared with the delay entailed by the initial molecule(Step 7 - Algorithm 2.6). If the delay of the initial molecule is greater than that of the newly generated ones, then the initial molecule is destroyed and the offspring of this reaction is added to the population (Step 9 - Algorithm 2.6). Otherwise, the newly generated molecules are destroyed (Step 11 - Algorithm 2.6) and the population remains the same.

Algorithm 2.6 Decomposition

Input: A molecule P_1
l the length of molecule P_1

Output: Two molecules P_2 and P_3
procedure Decomposition

1. Randomly select a node N_k from the molecule P_1
2. Generate a new path P' starting from node N_k to the destination node using Algorithm 2
3. Construct the molecule $P_2 = P_1\ [1 : N_k] \cup P'$
4. Generate a new path P'' starting from node the source node to node N_k using Algorithm 2
5. Construct the molecule $P_3 = P'' \cup P_1\ [N_k + 1 : \boldsymbol{l}]$
6. Calculate total delay for P_3
7. **if** delay(P_2) < delay (P_1) or delay (P_3)< delay (P_1)
8. Destroy the molecule P_1
9. Add P_2 and P_3 to the population
10. **else**
11. Destroy the molecules P_2 and P_3
12. **end if**
13. **end procedure**

The step-wise process to execute the cluster-based CRO meta-heuristics approach to solve the ILP is illustrated in Algorithm 2.7. The following steps are repeated until the maximum number of iterations (specified by network administrators) is attained. Molecules are generated as per Algorithm 2.2.

While the number of hits is smaller than the maximum number of iterations, two new numbers b and mol are randomly generated from the range [0;1]. If the value of molecule b is higher than that of molecule mol, then one molecule is selected from the set generated by Algorithm 2.2. If the number of hits is higher than the value of α (specified by network administrators as an input to the algorithm), then the decomposition process (Algorithm 2.6) is executed. Otherwise, the on-wall ineffective collision process (Algorithm 2.5) is executed. On the other hand, if the value of molecule b is lower than that of molecule mol, then two molecules are randomly generated and the amount of kinetic energy is decremented.

If the kinetic energy of the selected molecules is lower than the value of β (specified by network administrators as an input to the algorithm), then the synthesis process (Algorithm 2.3) is executed. Otherwise, the inter-molecular ineffective collision process (Algorithm 2.4) is executed. Finally, the number of hits is incremented and the solution minimizing the delay is computed and returned by the algorithm. It is worth stating here that the proposed algorithm, computes the optimal chaining between a pair of VNFs. As SFC requests usually tend to be composed of more than only two VNFs, we execute this algorithm in a parallel way between each pair of the same request, to ensure a sub-optimal solution for all the SFC components.

Algorithm 2.7 Min-Delay CRO

Input: Network size, Delay matrix *D[i] [j]*, Population size *PopSize,* Num -Hits= *0,* Maxlteration, Source node *S*, Destination node *D,*α= *0.6* * Maxlteration, β= *0.3* * Maxlteration
Kinetic energy *KE* = *Maxlteration*

Output: Final solution *O*

procedure CHEMICAL REACTION OPTIMIZATION

1. **while PopSize** has not yet been attained
2. Generate molecules by executing Algorithm 2
3. Randomly generate *mol* $\in$ [0,1]
4. **while** *Num- Hits* <*Maxlteration*
5. Generate ***b***$\in$ [0,1]
6. **if** ***b*** >*mol* **then**
   Randomly select one molecule
   if *Num-Hits* >α **then**
   Execute Algorithm 6
   else
   Execute Algorithm 5
   end if
   else
   Randomly select two molecules
   If *KE*<β **then**
   Execute Algorithm 3
   else
   Execute Algorithm 4
   end if
   KE --
   end if
7. Num-Hits++
8. **end while**
9. Check for a new minimum solution
10. **end while**
11. **Return** final solution and its objective function
12. **end procedure**

2.5.3.2 Genetic Algorithm optimization approach:

One of the most critical issues regarding chromosome encoding in the genetic algorithm framework is its feasibility, i.e. whether or not the decoded chromosome would result in a valid solution for the considered problem. As a first intuition, we used the same encoding presented in the CRO approach to implement classic genetic algorithm (C-GA).

Nonetheless, what limits the performance of C-GA is the condition of the size imposed on the generated paths as specified in the GA framework; this size should remain the same from parents to children, and this limits the search space and increases the probabilities of having the algorithm stuck in a local optimum. Moreover, if the initial solution did not contain a path with the exact same size as the optimal solution, the algorithm would never converge to the optimal, and would be stuck on near-optimal or even bad solutions. To avoid this problem, we decided to adopt a priority-based encoding. The proposed encoding uses the chromosome not to represent the solution (path) itself, but rather, assigns for each node a priority that would help in constructing the path. The size of chromosomes with this encoding is set to the number of nodes in the initial graph, and any permutation-based operator would work for crossover and mutation since the order of genes would not affect the feasibility of the decoded solution. Algorithm 2.8 is introduced to generate priority values for the nodes in the substrate network. This is done by randomly choosing a number between 1 and N, with N being the size of the network.

Algorithm 2.8 Genetic Algorithm-Chromosome generation

Input: Network size N

procedure GENERATE CHROMOSOMES

1. **For i=0 to N**
2. P[i] : a random number in the range [0,N] representing the priority value of node i.
3. **end for**

Algorithm 2.9 illustrates the crossover process of the priority based GA. Crossover is a genetic operator that is based on merging the genetic information of two parents to produce new offspring. In other words, it is useful for stochastically producing new solutions from an existing population. In the first step of Algorithm 2.9, a random cut point n is applied on two input parent chromosomes (Step 1 - Algorithm 2.9). A new child chromosome C3 is then created by partially populating its priority values with the priority values of one parent C1 from the index range [1;n] (Step 2 -Algorithm 2.9). The remaining priority values are populated from the second parent chromosome C2 such as to not overlap with the priority scores obtained from the first parent.

The same process is repeated to create another child chromosome C3, while changing the order of the populating chromosomes (i.e., starting from C2 then going on to C1) (Step 5-Algorithm 2.9). Finally, the number of hits is incremented after each crossover operation. Algorithm 2.10, explains the mutation process of the priority based GA is explained. Mutation simulates the biologic mutation phenomenon and is important for preserving diversity from one generation of chromosomes to the next. The input to Algorithm 2.10 is a parent chromosome C1 and the output is a child chromosome C2. The mutation process is implemented by randomly selecting two node indexes n1 and n2 from the parent chromosome and then swapping their priority values. Finally, the number of hits is incremented after each mutation operation. The decoding process is illustrated in Algorithm 2.11.

The inputs to the algorithm are a chromosome C, a source node, a destination node, and an adjacency matrix recording each node’s neighbors list. The output is a decoded chromosome obtained after some manipulations. The first step consists in determining the source node’s neighbor having the highest priority value and storing it into chromosome P. This process is repeated until the intended destination node is reached and the full decoded path P is formed.

Algorithm 2.9 Priority-based Genetic Algorithm Crossover

Input: Two parent paths C_1 and C_2

Output: Two child paths C_3 and C_4

procedure Crossover

1. Randomly choose a cut point *n* on both C_1 and C_2
2. Copy the priority values from $C_1[0]$ to $C_1[N]$ into C_3
3. **while** *length*(C_3) $<N$ **do**
   for $i = 0$ **to** *length*(C_2)
   if $C_2[i] \notin C_3$
   Add $C_2[i]$ to C_3
   end if
   end for
4. **end while**
   Copy the priority values from $C_2[0]$ to $C_2[N]$ into C_4
5. **while** *length*(C_4)$<N$ **do**
   for $i = 0$ **to** *length*(C_1)
   if $C_1[i] \notin C_4$
   Add $C_1[i]$ to C_4
   end if
   end for
   end while
   Increment *Num-Hits*++
6. **end procedure**

Algorithm 2.10 Genetic Algorithm-Mutation

Input: One parent path C_1

Output: One child path C_2

procedure Mutation

1. Randomly select two node indexes $n_1$and n_2 from C_1
2. Swap priority values between the selected nodes
3. *Num-Hits*++
4. **end procedure**

Algorithm 2.11 Genetic Algorithm Decoding

Input: Chromosome C, S_i. set of node *i*'s neighbors, Source node *src_node,* Destination node *dst_node*

Output: Decoded chromosome *P*

```
procedure Decoding
        i = src_node
        k = 0
        n = max{C[i], i∈ Si}
        P[k] = n
        k++
        while P[k]! = dst_node do
           i = P[k-1]
           n= max{C[i], iE Si}
           if n∉ P
              P[k] = n
              k++
           else
              Si= Si\n
           end if
        end while
end procedure
```

Algorithm 2.12 describes the stepwise methodology to execute the different steps of the priority-based GA process. First, a number of chromosomes that is equal to the desired population size is generated using Algorithm 2.8 (Step 2 - Algorithm 2.12). These

chromosomes then undergo the decoding process using Algorithm 2.11 (Step 3 - Algorithm 2.12). The delay of the decoded chromosomes is compared with the minimum tolerable delay decided by network administrators as an input to the algorithm. If the delay of the decoded chromosomes falls below the minimum tolerable delay, the decoded chromosomes are retained and are added to the population. Otherwise, the chromosomes are discarded (Steps 4- 5 - Algorithm 2.12). Then, the following steps are repeated as long as the number of hits (which is updated after each mutation and crossover process) is lower than the maximum number of iterations specified as an input to the algorithm. A number is randomly generated (Step 9 - Algorithm 2.12). If the generated number is lower than the crossover rate (an input to the algorithm), then two chromosomes are randomly selected (Step 10 – Algorithm2.12), the crossover process is executed on those chromosomes using Algorithm 2.9 to generate two new child chromosomes, the decoding process is executed on the parent and child chromosomes using Algorithm 2.11, and the delay entailed by the child chromosomes is compared with those entailed by the parent chromosomes.

If the delays of the child chromosomes are lower than those of the parent chromosomes, the child chromosomes are retained in the population and the parents are destroyed. If the randomly generated number in Step 15 is lower than the mutation rate (an input to the algorithm), then one parent chromosome is randomly, the mutation process is applied on this chromosome using Algorithm 2.10 to generate a child chromosome, the decoding process is applied on both parent and child chromosomes, and the delay entailed by both the child chromosome and parent chromosome are compared. If the delay of the child chromosome is lower than that of the parent chromosome, the child chromosome is retained in the population and the parent is destroyed.

Algorithm 2.12 Min-Delay Priority based GA

Input: Network size**,** Delay matrix *D[i] [j]*, Population size *PopSize,*
Num -Hits = 0, *Maxlteration,* Source node *S,* Destination node *D,*
Mutation rate *MuRate* = 0.2, Crossover rate *CrossRate* = 0.8,
SFC minimum tolerable delay D_s

procedure Chemical Reaction Optimization

1. **while** *PopSize* has not been reached yet **do**
2. Execute Algorithm 8 to generate a chromosome and store it in *P*
3. Execute Algorithm 11 with *P* being the input chromosome to generate the decoded chromosome *P′*
4. **if** *delay(P')* < D_s **then**
5. Add *P* to the population
6. **end if**
7. **end while**
8. **while** *Num-Hits* <*Maxlteration*
9. Randomly generate r ∈ [0,1]
10. **if** *r* <CrossRate **then**
    Randomly select two chromosomes $C_1'$and C_2'
    Execute the crossover on $C_1'$and C_2' using Algorithm 9 togenerate two child chromosomes $C_1''$and C_2''
    Execute the decoding process on the parent chromosomes$C_1'$and $C_2'$and child chromosomes $C_1''$and C_2''using Algorithm 11
        if *delay*(C_1'')< *delay*(C_1'') or *delay*(C_2'') < *delay*(C_2')
        Add $C_1''$and $C_2''$to the population and destroy $C_1'$and C_2'
        end if
11. **end if**
12. **if** r <MuRate **then**
    Select one parent chromosome *C′*
    Execute the mutation process on *C′* using Algorithm 10 to generate another child chromosome *C″*
    Execute the decoding process on the parent chromosome *C′* and child chromosome *C″* using Algorithm 11
        if *delay(C″)* < *delay(C')***then**
        Add *C″* to the population and destroy *C*
        end if
13. **end if**
14. **end while**
15. **Check for a new optimal solution**
16. <= final solution and its objective function
17. **end procedure**

2.5.4 CRO Based Placement Algorithm:

In this section, we will explain how we employed the CRO algorithm to solve our ILP for the second objective function. We considered CPU for our implementation, but it can be easily adapted to optimize the problem in terms of other metrics. The objective of Algorithm 2.17 is to provide a set of solutions for the placement phase that ensure a high consolidation of physical resources on the hosting nodes, which thus means a minimum number of servers will need to be used to host the incoming requests.

2.5.4.1 Solution Representation and Molecule generation:

The molecule encoding for this part of the algorithm is represented in Figure 2.3. The placement phase is similar to the bin packing problem, which is why we adopted a grouping technique to encode our molecule. Each group represents a server and the content of the groups represents the indexes of their hosted VNFs. The index of each VNF is specified in the SFC request like illustrated in Figure 2.2. To generate a set of initial solutions, we adopted the First Fit (FF) algorithm. The main idea underlying this heuristic is that for each VNF, the FF algorithm attempts to place it in the first server that can still host it based on its residual CPU capacity. If all the used servers are full, the algorithm starts up a new server and puts the VNF in it. To make sure that we have different solutions in the initial population, the order for placing the VNFs is chosen randomly at each molecule generation. Once we generate our initial population, those molecules will undergo the elementary reactions defined by the CRO framework to find a near optimal solution.

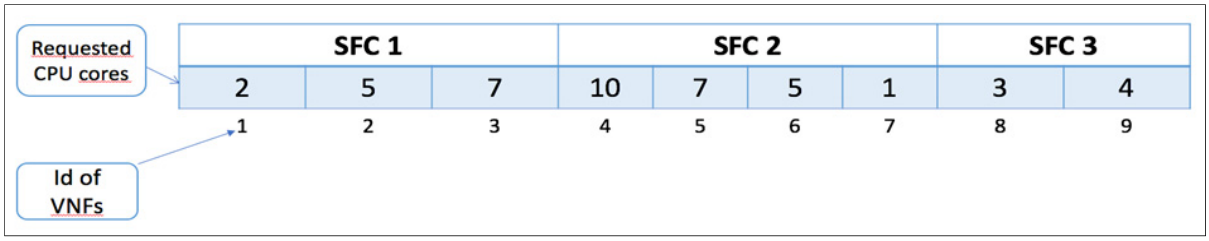


Figure 2.2 Input Requests

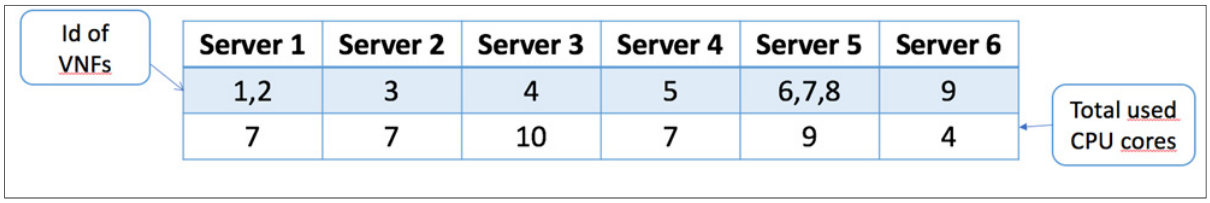


Figure 2.3 Molecule Encoding

Algorithm 2.13 CRO Placement Molecule Generation

Input: SFC Request, Servers Capacity, Count=Request.size
Output: P: Placement Solution
procedure Generate Molecules
While count !=0
Pick a random VNF V from the SFC request.
if (V is not already placed)
Apply first fit algorithm to host V based on the servers' available capacities
Update the hosting server's capacity
Count--
else
Pick another VNF from the request
end while
Return P

2.5.4.2 Elementary Operation for CRO:

Decomposition:

This procedure results in two molecules C1 and C2 from a molecule P chosen from the current population. In this elementary reaction, we maintain the server with the maximum used capacity from P and we copy its content into two new solutions. The remaining VNFs that are not contained in this server are placed again using the first fit algorithm on both new solutions. The order adopted for placing the remaining VNFs is random on C1 and C2, to ensure that the offspring is different. The procedure for this reaction is described in Algorithm 2.14.

Algorithm 2.14 CRO Placement Decomposition

Input: One Molecule P_1
Output: Two Molecules $C_1$and C_2

procedure Decomposition

for each server inP
Calculate the sum of allocated CPU
Rank the servers in a descending order based on their used CPU
end for

Choose the server with maximum used CPU S_1
Initialize a new solution C_1
Copy the content of S_1in the first server of C_1
Reallocate the remaining VNFs in the new solution C_1 by using the first fit algorithm //The order of the VNFs to be placed is random
Initialize a new solution C_2
Copy the content of S_1 in the first server of C_2
Reallocate the remaining VNFs in the new solution C_1 by using the first fit algorithm
if Fitness(C_1) > Fitness(P) **Or** Fitness(C_2) > Fitness(P)
Remove P from population
Add $C_1$and C_2 to the population
else
Discard $C_1$and C_2
end procedure

On-wall ineffective collision:

This procedure represents a corrective action where we attempt to improve the quality of a selected solution. Unlike decomposition, this elementary reaction starts with one molecule P and results in one molecule C. The main idea is to choose the weakest server S_1 on solution P with the least used CPU capacity. The next step is to empty the selected server S_1 and redistribute its VNFs on the currently used servers, as they may still have enough CPU available to host the VNFs previously hosted on S1. The procedure for this reaction is described in Algorithm 2.15.

Algorithm 2.15 CRO Placement On Wall Ineffective Collision

```
Input: One Molecule P
Output: One molecule C
procedure ON-WALL COLLISION
Copy the solution P into C
For each server inC
    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 S1
Empty the selected server S1
Redistribute the previously hosted VNFs on S1using first fit algorithm
  if Fitness(C)> Fitness(P)
    Remove P from population
    Add C to the population
  else
    Discard C
end procedure
```

Synthesis:

This procedure results in one solution C starting from two molecules P1 and P2. The aim of this elementary reaction is to copy the best features from selected molecules P1 and P2 into the offspring C. The first step is to look for the fullest servers on both solutions P1 and P2 and then copy their content into the offspring solution C. Before proceeding with the rest of the VNFs, it is mandatory to remove potential VNF duplicates in S1 and S2. Once we are sure of the unicity of placed VNFs, we host the remaining VNF requests using the first fit algorithm. Details of the procedure are explained in Algorithm 2.16.

Algorithm 2.16 CRO Placement-Synthesis

```
Input: Two Molecule P1 and P2
Output: One molecule C

procedure On-wall Collision
    for each server in P1
        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 S1
    endfor

  for  eachserver in P2
        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 S2
  end For

  Initialize a new empty solution C
  Copy the placed VNFs in S1 and S2 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 Fitness(C)>Fitness(P1) Or Fitness(C) > Fitness(P2)
      Remove P1 and P2from population
      Add C to the population
  else
      Discard C
end procedure
```

Inter-molecular ineffective collision:

This procedure starts with two molecules P1 and P2 and results in two solutions C1 and C2. Since each new solution is constructed independently from the other, we use the on wall ineffective operator twice to generate new solutions. C1=ON-WALL COLLISION(P1) and C2=ON-WALL COLLISION(P2).

Fitness function:

As our aim in the placement phase is to optimize the operational cost, we must maximize the hardware resource consolidation and use the minimum number of servers to host all incoming requests. That's why we define our fitness function for CRO based placement by Eq.(2.13). Maximizing this fitness function along the CRO iterations allows favoring the most-filled servers and thus avoids hardware resource waste by having the maximum number of inactive servers.

$$Fitness_{placement} = \frac{1}{|V_h|}\sum_{n=1}^{|V_v|}\sum_{m=1}^{|V_h|}\left(\frac{\alpha_{n,m}^{p}.R_m^p}{R_{Vh}^{max}}\right)^2 \tag{2.13}$$

The stepwise process to execute the CRO-based placement approach to solve the ILP in terms of operational cost and CPU consolidation is illustrated in Algorithm 17.

Algorithm 2.17 CRO based placement

Input: Network size, Scap: Servers' capacities, SFC requests, Num-Hits = 0, Maxlteration, Popsize: Initial population size, α= 0.6 * Maxlteration, β=0.3 * Maxlteration, Kinetic energy KE = Maxlteration

Output: Final solution **O**

Procedure CHEMICAL REACTION OPTIMIZATION

While Popsize has not yet been attained
Generate molecules by executing Algorithm 13
Randomly generate mol∈ [0,1]
while Num-Hits <Maxlteration then
Generate b∈ [0,1]
If b >mol
Randomly select one molecule
If Num-Hits >α then
Execute Algorithm 14
else
Execute Algorithm 15
else
Randomly select two molecules
If KE <β
Execute Algorithm 16
else
Execute Algorithm 15 on both selected molecules
end if
KE- --
end if
Num-Hits++
end while
end while
Check for a new minimum solution
O <= final solution and its objective function
end procedure

2.6 Experimental Setup and Results:

In this section, we provide the details of the used environment, the conducted scenarios and we discuss the obtained results. We classify our simulations into two separate sets:

- The first part consists of testing the chaining phase to decide which of the heuristics presented in section 4.3 to adopt and also determining the limit of network size to decide when to apply clustering.
- The second part of our simulations shows the overall results for our proposed approach for VNF placement and chaining.

2.6.1 SFCs chaining without clustering:

We consider a substrate network composed of 30 nodes that are either middleboxes or host servers. The delay on the edges of the network is set randomly between [30,250] us, while the SFC request tolerates a maximum delay of 450us between two adjacent VNFs. Our proposed algorithm returns the optimal path between a pair of VNFs that we assume, in a first step, are already placed.

Heuristic parameters:

- For the classic version of GA (C-GA), we used a high rate for mutation (0.9) as it is more likely to improve the offspring than crossover which is limited to the condition of having a crossing point between the two parents to ensure having a valid solution (Crossover rate 0.1).
- For the priority-based GA (P-GA), we used the default settings for mutation and crossover rates which are respectively 0.2 and 0.8;
- For CRO the parameters were set based on a trial and error process: α=0.6×MaxIteration, β=0.3×MaxIteration

To define the maximum number of iterations needed to find optimal solutions for each of the proposed algorithms, we ran them on the same substrate graph on which we already know the optimal path from a source to destination node. Figure 4 shows the evolution of the End-to End (E2E) delay when the algorithm is run a single time. The priority-based GA (P-GA) converges to the optimal solution in less iterations than CRO and the classic GA. C-GA, and

because of the limitations explained in section 2.5.3.2 gets stuck in a local optimum at around 300 iterations and might not converge to the optimal path if the quality of the initial population is not good enough. Moreover, since the path generation procedure is totally random at first, the probability of not finding a good solution increases for CGA. On another hand CRO manages to find the optimal path with an average of 350 to 400 iterations.

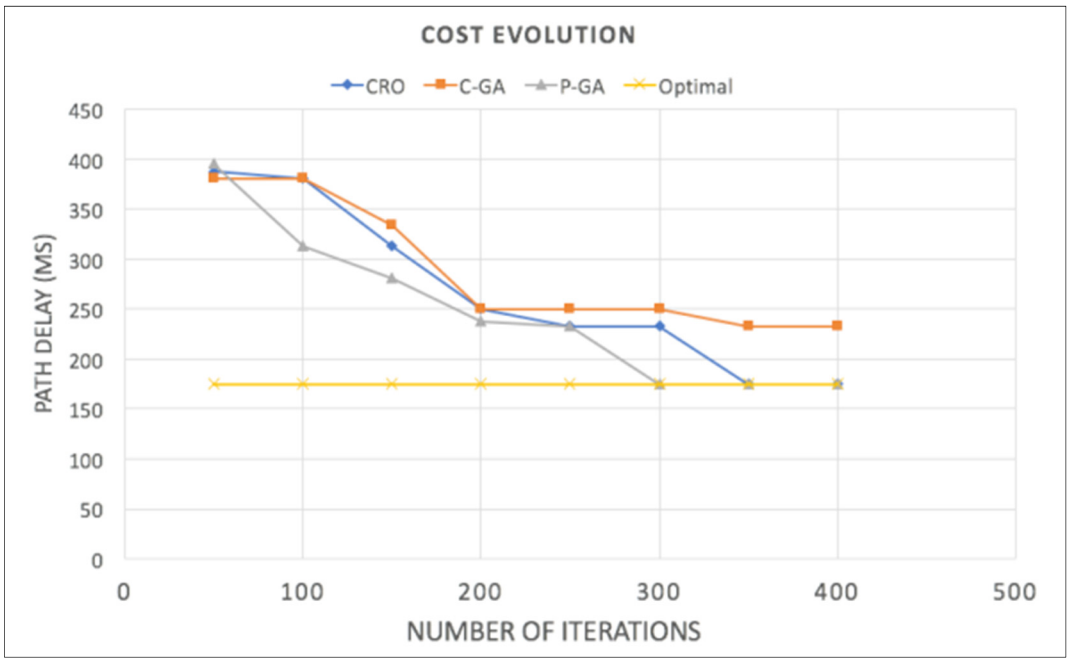


Figure 2.4 Comparison of E2E evolution for CGA, PGA and CRO

Now, to evaluate the performance of the proposed algorithms, we rely on two metrics, the quality of the solution in terms of the E2E delay as well as the run time needed to find a near optimal solution.

E2E Delay:

Based on the previous results of Figure 2.4, we fixed the iteration number for both CRO and CGA to 400 while for PGA we set it to 300. We run each of the algorithms 10 times to compute the E2E delay as well as the average execution time needed for each approach. The

considered network topology is always composed of 30 nodes (middleboxes and servers included). In Figure 2.6 we present the best, the worse and the average solution cost for each of the algorithms. As we can see that in the best case, all algorithms manage to find the optimal path. As we consider the average cost delay, the worst performance is that of C-GA and that is due to the sizing condition imposed on the chromosomes. Because, if the initialization did not contain a path of the exact same size of the optimal one, the C-GA algorithm would never be able to converge to better solutions. CRO and P-GA perform similarly in terms of the solution quality but as we mentioned earlier, since the run time is a critical criterion when it comes to latency critical applications such video conferencing and voice-over-IP, we compare our solutions also in terms of execution time.

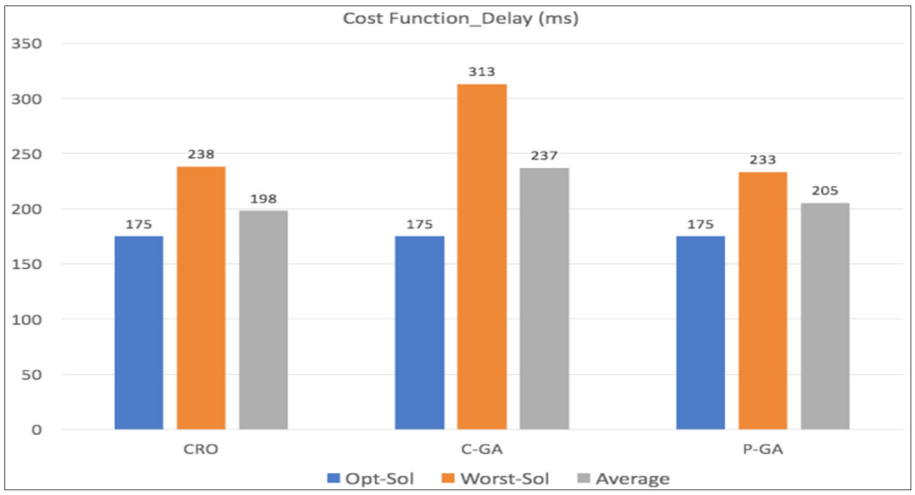


Figure 2.5 Average E2E Delay for C-GA, P-GA and CRO

Run time:

The number of needed iterations is not directly correlated with the run time as we can see in Figure 2.7 showing a comparison of the execution time for the three studied algorithms (P-GA, C-GA and CRO). Based on the obtained results, we can see that C-GA have the shortest execution time compared to CRO and P-GA but as previously explained is more likely to result in bad solutions. On another note, P-GA takes more time to converge to optimal compared to CRO, even if he needs less iterations. This is an expected result as the P-GA procedure requires an extra processing phase, which is to decode the chromosome into a valid path, while for CRO a molecule represents actually a valid solution we don't need another procedure to transit from the search to solution space.

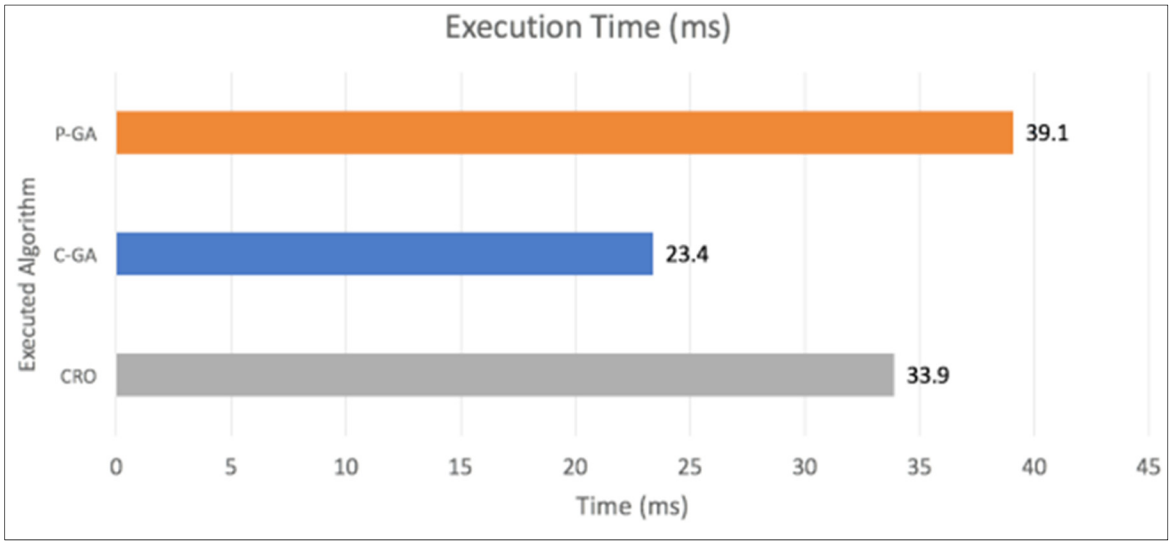


Figure 2.6 Average run-time for C-GA, P-GA and CRO

According to the obtained results both in Figures 2.6 and 2.7, it is obvious that the C-GA is not the heuristic to adopt for our chaining problem, which led us in the following simulations to compare just CRO and P-GA.

To decide which of the two algorithms to adopt, we doubled the substrate graph size to 60 nodes to see how this parameter impacts the efficiency of our proposed solutions. Here again

we run each one of the algorithms 10 times with the same parameters used for the previous simulation.

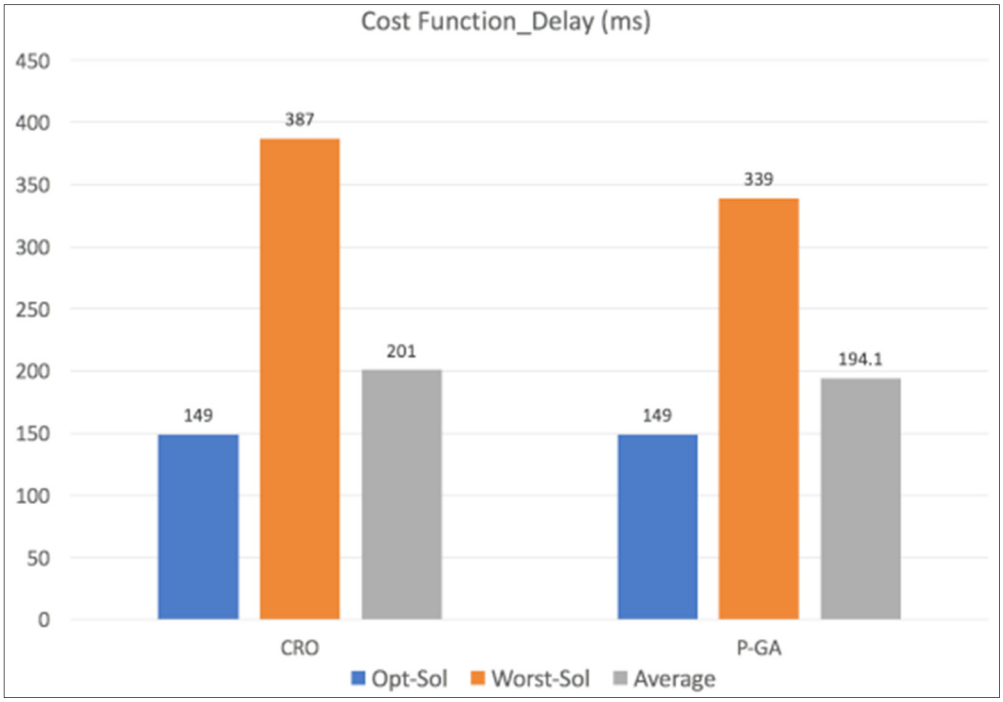


Figure 2.7 Average E2E delay for P-GA and CRO

To evaluate the run time, we increase the size of the network from 20 to 120 nodes. We can conclude based on results of Figure 2.8 that the average E2E delay found by P-GA is slightly better than that of CRO. But when considering the execution time, we can clearly see in Figure 2.9 that as we increase the size of the input graph, the execution time increases also for both approaches but the gap between CRO and P-GA gets more important. Given the fact that we consider time sensitive applications and based on our results, we decided to adopt the CRO for the chaining phase of our overall approach for SFC embedding.

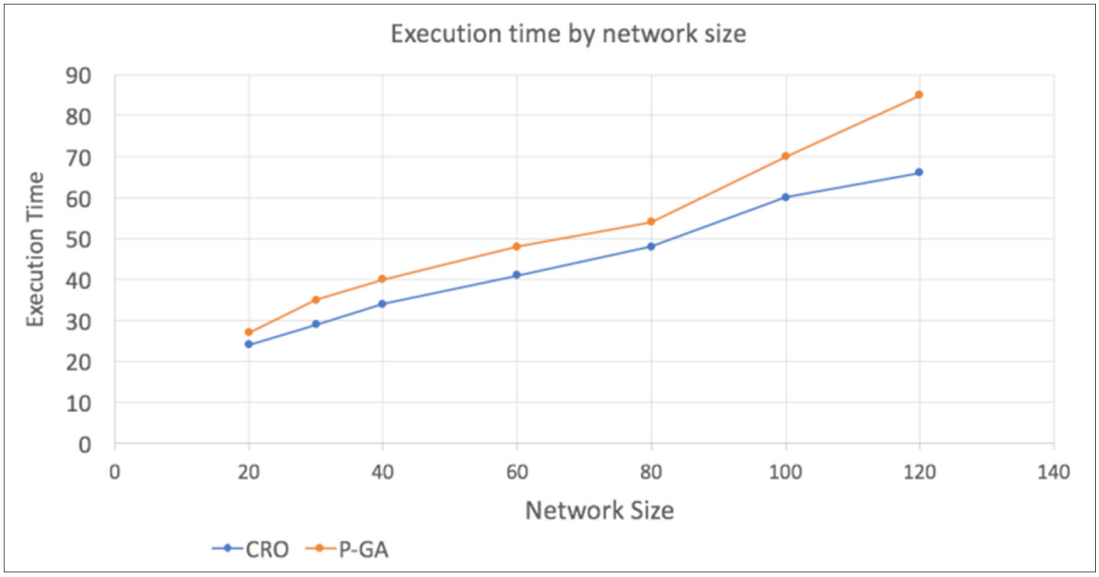


Figure 2.8 Average Runtime for PGA and CRO by network size

The proposed CRO based chaining finds the optimal path for a pair of VNF, but in real case scenarios SFCs are usually composed of more than just two VNFs, that's why we parallelized the proposed procedure in Algorithm 2.7 to find the optimal path between each pair of VNFs composing the same SFC. We tested our parallelized algorithm using the parameters presented in Table 2.1.

Table 2.1 Simulation Setup for chaining

Simulation parameters for chaining	
Total SFC requests	20
SFC size	2 to 10 VNFs
E2E delay requirements	Set randomly between [200,500] us
Delay on substrate links	Set randomly between [30,250] us
Substrate network size	30,60,90,120

As we can see in Figure 2.9, for the same number of VNFs in the SFC requests, the E2E delay increases as we increase the size of the network. It is an expected result as there are no restrictions on where to place the VNFs which means that a pair of VNFs are not mandatorily placed near each other as we increase the size of the network.

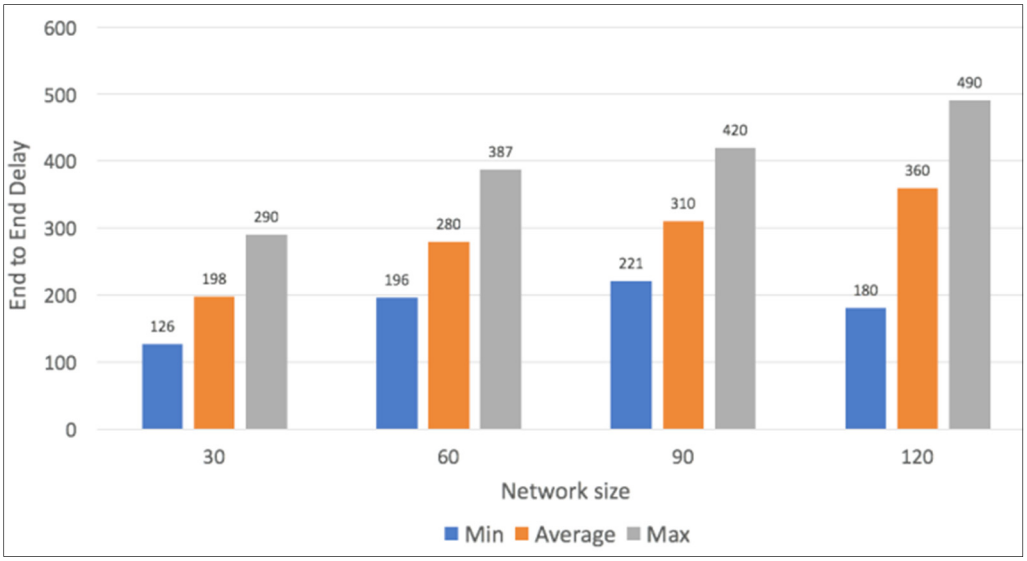


Figure 2.9 Average E2E Delay by network size for parallel CRO

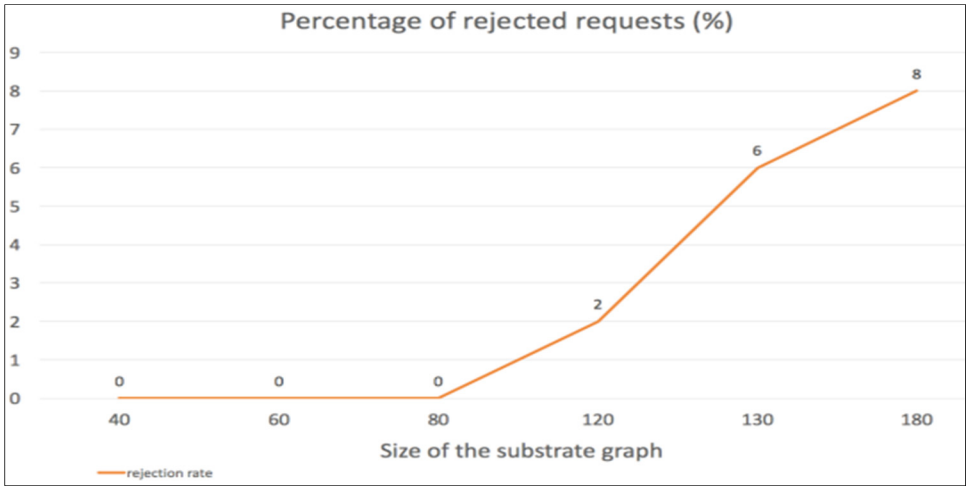


Figure 2.10 Rejection rate of SFC requests

We can see in Figure 2.10 that not only does the E2E delay increases but also the rejection rate of chaining a given SFC. Even if the network has enough physical resources to host all the VNFs but in terms of latency the probability of violating the delay requirements of SFCs keeps increasing as we consider larger substrate networks.

This result is explained by the fact that all SFC requests are placed simultaneously, which increases the probability of having VNFs belonging to the same SFC but hosted on extreme nodes in the substrate graph. Moreover, if the link delays are too high the request would not be fulfilled in terms of E2E delay and would thus be rejected. To avoid having a high rejection rate, clustering can be a helping process to reduce the placement and chaining search space. For a clustered network, using the same approach we can make sure that the whole SFC is hosted on a relatively small network, which guarantees that its components would be closer to each other and thus avoid violating the SFCs' latency requirements.

2.6.2 SFCs Placement and chaining with clustering:

The simulations in this section were conducted on a 120 nodes substrate graph, this choice is based on the obtained results in Figure 2.10 as the rejection rate starts getting higher starting from 80 nodes which is the limit of the network size to apply clustering.

Clustering:

Using the k-medoids based algorithm we form a set of energy efficient clusters as well as memory efficient clusters. The parameters used to simulate our clustering are presented in Table 2.2.

Table 2.2 Simulation Setup for clustering

Simulation parameters for clustering	
Number of nodes in physical graph	120
Available energy per node	Set randomly between [3000,5000] Joules
Available memory per node	Set randomly between [100,2000] Go

In Figure 2.11, we can see how the clustering process classifies the substrate nodes based on their energy capacities. Each cluster stands for a level of efficiency here in terms of energy and in terms of memory in Figure 2.12. The SFC requests are then assigned to the according cluster, based on their preferences whether they prioritize energy or memory and their requirements for each of these metrics.

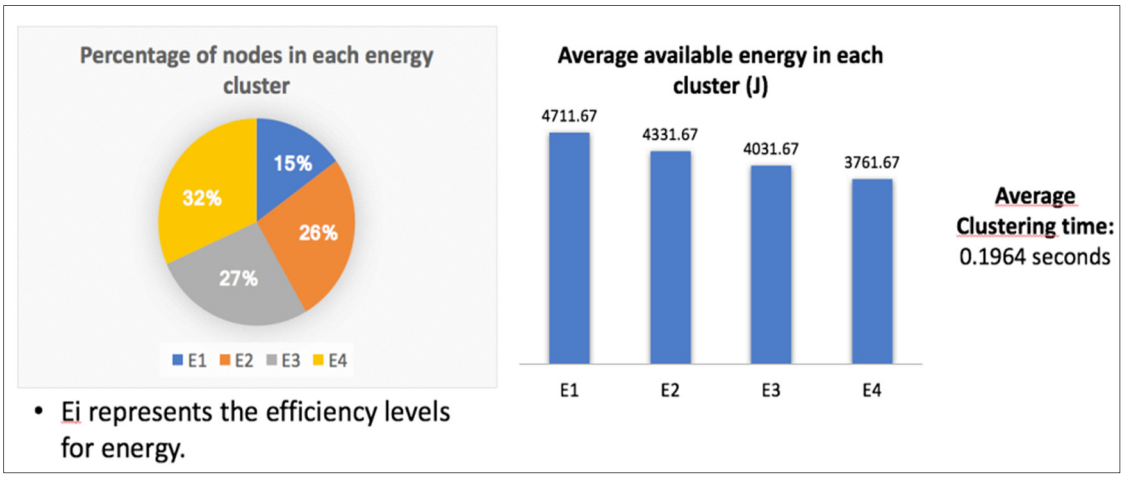


Figure 2.11 Node distribution according to Energy availability

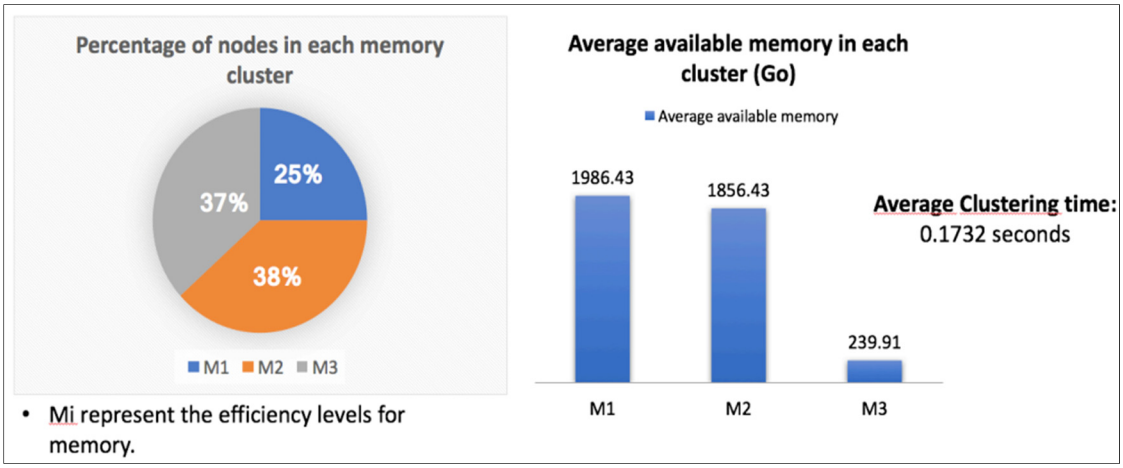


Figure 2.12 Node distribution according to Memory availability

Placement:

We compare here the results of CPU consumption and resource consolidation in two network scenarios: with and without clustering. To simplify our simulations, we consider that all VNFs are of the same type, meaning they request the same type of clusters (energy or memory efficient). The servers' capacity is also set to the same value but in realistic environments these capacities are dynamically monitored and given as input to our placement algorithm. The used parameters are described in Table 2.3.

Table 2.3 Simulation setup for placement

Simulation parameters for placement	
Number of nodes in physical graph	120
Number of clusters	4
Servers' initial capacity	30 cores per server
Number of SFC requests	15,20,30,40
SFC size	2 to 10 VNFs
VNF CPU core requirements	1 to 15 cores

Figure 2.13 shows the comparison of used servers when the network is considered without clustering and in the second case when we apply the clustering.

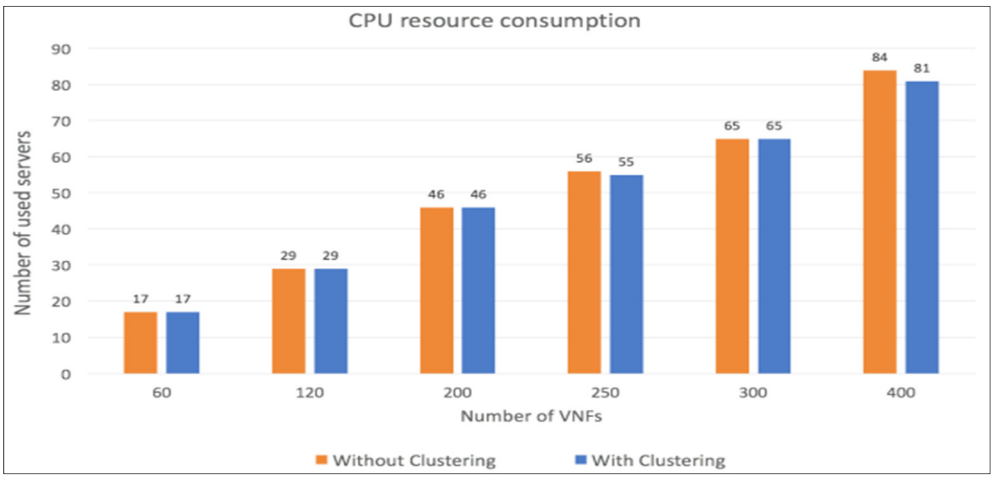


Figure 2.13 Comparison of number of used servers

Based on the results, we can see that up to a total of 120 VNFS only one cluster is enough to host the requests and the total number of used servers is the same in both scenarios. As we increase the number of requested VNFs to 200, we can see in this case that 2 clusters are needed to fulfill the SFC requests and the total number of used servers starts to improve slightly starting from 250 VNFs as a total request. As we increase the SFC request, the clustering allows us to reduce the number of used servers and thus to consolidate our physical resources. This result is explained by the fact that the placement algorithm does not move to a second cluster unless the 1st one is fully used, while if the whole network is considered the placement algorithm has a larger search space and might place the requests not necessarily in the most optimal way.

Chaining:

In this part of the simulation we fix the total number of VNFs to be hosted to 200, and compare the quality of the CRO solutions in both cases: with and without clustering the substrate network.

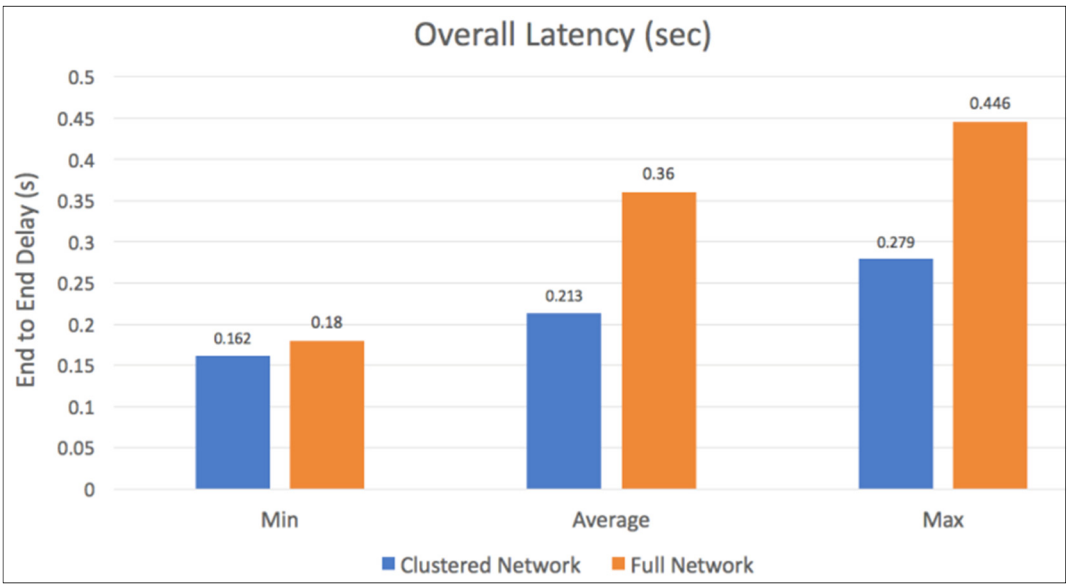


Figure 2.14 E2E delay comparison with and without clustering

We can see based on Figure 2.14, that the average E2E delay found in the case of a network without clustering is considerably higher than that of a clustered substrate network. This result is expected principally due to the placement phase, as we place SFCs on their dedicated cluster, this increases the probability of having VNFs belonging to the same SFC hosted relatively close to each other and thus resulting on a better E2E delay. As we consider the worst-case scenario, we can see that for a network without clustering, the E2E delay is practically twice the case where the network is clustered. Not only does the clustering helps improve the quality of the SFCs chaining, but it also ensures that all the requests are placed and chained according to their initial requirements, contrary to the case where we consider

the whole network as we've seen in Figure 2.10 the rejection rate keeps increasing as we consider larger substrate networks.

Execution time:

After we've analyzed how the clustering technique helps improve the quality of the proposed solutions, in this section we will analyze its impact on the overall run time of the proposed approach. In the Figure 2.15, we compare two schemes of clustering:

- Static clustering where its executed at the start of the algorithm and not updated again, in this case it wouldn't have a big impact on the overall execution time.
- Dynamic clustering updated periodically as the resources vary on the network topology, in this case it should be considered in the execution time.

For both cases and based on the results of Figure 2.15, we can clearly see that the clustering technique improves considerably the run time of both placement and chaining, that is due mainly to the fact that the search space is reduced and therefore the complexity of the problem also.

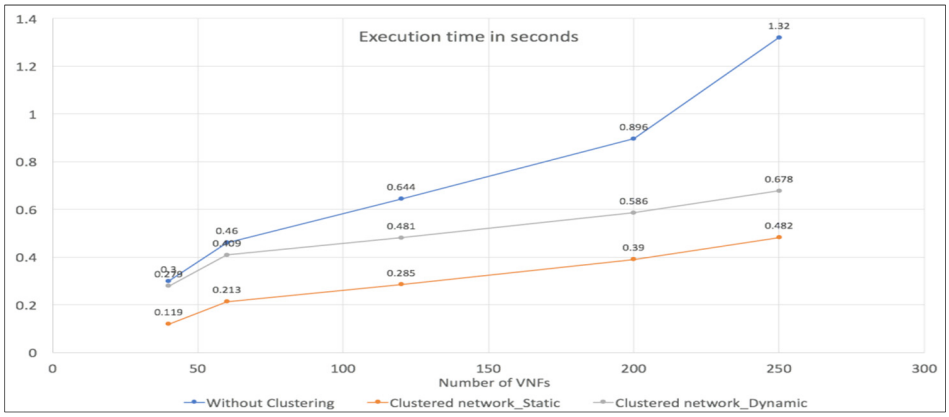


Figure 2.15 Run-time comparison of placement and chaining with and without clustering

2.6.3 Results analysis:

The following section summarizes the results for the different stages of our approach:

Chaining:

Table 2.4 presents our obtained results for a substrate graph composed of 30 nodes, based on the obtained results we can see that the C-GA performs bad compared to the two other approaches: P-GA and CRO, which made us discard this solution. Considering execution time also as an important factor we can see that even CRO and P-GA might perform similarly in terms of solution quality, the run time is better for CRO. As we increased the number of nodes to 60 we can see as presented in Figure 2.9 that the gap keeps increasing as the network size increases so we decided to adopt CRO for our chaining phase.

Table 2.4 Heuristic comparison for chaining

Heuristic	Average E2E Delay (ms)	Execution time (ms)
CRO	198	33.9
P-GA	205	39.1
C-GA	237	23.4
Optimal	175	--

Placement and chaining:

Table 2.5 presents our obtained results for both placement and chaining as we consider two case scenarios: a substrate network without clustering and the second case is where we apply the K-medoids clustering approach. The heuristic used in this phase is CRO for both stages: placement and chaining, while the substrate network is here composed of 120 nodes.

Table 2.5 SFC embedding performance comparison

	Average E2E Delay (sec)	**Average number of used servers**	**Execution time (sec)**
With Clustering	0.213	81	0.57
Without Clustering	0.36	84	1.32

As previously explained and following the presented results, we can see that the clustering phase helps not only to include other performance metrics (e.g. energy, memory, storage...) but can help optimize the execution time of the embedding procedure and also improve the quality of the resource consolidation and the E2E delay values.

2.7 Conclusion

In this chapter, we propose a multi stage approach to ensure an optimized placement and chaining of service function chains by minimizing the operational, transmission and penalty costs. Our formulated approach uses the heuristic algorithm CRO to find sub optimal solutions, it also offers many advantages compared to the proposed solutions in the literature as it:

- Takes many metrics into account in the optimization process more specifically energy and CPU consumption as well as the end to end delay. It is worth stating here that the approach is generic enough to be adapted to consider other metrics depending on the priorities of the SFC requests.
- Scales well with large network topologies, thanks to the pre-processing phase of clustering that helps reducing the space search and thus the complexity of the problem.
- Acts in a dynamic scheme as we update the clustering periodically depending on the resources and depending on the incoming SFC requests and their resource consumption. It allows network administrators to decide of the most efficient placement and adjustment policies as the resources in the cloud datacenters are constantly changing.

Our simulations show that our approach proves to be efficient for the SFCs placement and chaining problem. Using the CRO, results prove that this heuristic outperforms genetic algorithm especially in the chaining phase as it offers a good trade-off between the quality of its solutions as well as the run-time. Results shows also that Clustering help not only to include other objective functions making the placement more optimized, but it also helps reducing the space search for our adopted heuristic. Clustering allows also optimizing the run time of the whole process, which is an important factor as we consider time sensitive applications and real-time services like in 5G networks. Consolidating resources on the substrate level and also ensuring this efficient chaining between VNFs allows optimizing the operational and penalty costs we defined in section 2.4

As future work, we plan on extending our work in two principle axes:

- Investigating other clustering techniques to improve the quality of the formed clusters and also the execution time.
- Defining a joint placement and chaining approach for SFCs using metaheuristic algorithms, which will also allow us to improve the overall execution time of the procedure along with ensuring an optimal tradeoff between the set of metrics we want to optimize. (e.g energy, delay, CPU, memory...)

CHAPTER 3

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

Imane El Mensoum[a], Laaziz Lahlou[b], Fawaz A.Khasazneh[c], Nadjia Kara[d], Claes Edstrom[e]

[a b c d]Department of Software Engineering and IT, École de Technologie Supérieure, 1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[e] Ericsson Canada, 8275 Route Transcanadienne, Ville Saint-Laurent, Québec, Canada H4S 0B6

Paper Submitted to the journal "IEEE Transaction on Cybernetics" in October 2019

Provisional patent application P79104 filled by Ericsson Patent Unit Canada to the United States Patent and Trademark Office (USTPO) in October 2019

3.1 Abstract

Complex networks 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. Nowadays, computer networks are undergoing significant expansion due to the proliferation of network devices (legacy and/or virtualized), compute nodes and the number of relationships/inter dependencies between network nodes. One of the main challenges in computer communication networks lies in categorizing these compute nodes into communities/clusters and detecting clusters/communities of connected compute nodes within these large-scale structures sharing similar features of different types (e.g., CPU, memory, disk, etc.).

In this paper, we propose a set of novels, dynamic and proactive topology-aware unsupervised machine learning (i.e., clustering) approaches, namely, a mixed integer linear

program, a chemical reaction optimization meta-heuristic and a game theory approach, 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 Network Function Virtualization and Cloud based Communication Networks. In this regard, the solutions aim to help decision makers facing issues related to scalability and computational complexities of their mechanism to deploy their cloud- and non-cloud-based virtual network services in an effective manner.

To the best of our knowledge, we are the first to consider the problem in the above contexts and to propose a solution. Experimental results demonstrate the effectiveness of the proposed approaches and their suitability, given their polynomial time complexities, which make them easy to deploy and integrate into cloud providers' orchestration systems.

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

3.2 Introduction

Complex networks (Bothorel et al. (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 (Alessandro et al. (2017), Yang et al. (2009), Hong et al. (2011), Chen et al. (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 (Yang 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, 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), Yang et al. (2009), Alessandro et al. (2017), and Hong 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 both structural and node properties.

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

- 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, 5G and network slicing.
- 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.
- 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 benchmarking.
- A game theory-based approach similar to item 3 above is also suggested for forming clusters.
- Moreover, our proposed solutions may be integrated into orchestration systems following NFV MANO thanks to their low computational complexities.
- VALKYRIE's performance in terms of solution quality and scalability is assessed using real-world topologies: small, medium and large-scale enterprise networks.

The rest of this chapter is organized as follows. Section 3.2 presents the context, motivation and system architecture. The related works is discussed in Section 3.3. Section 3.4presents the system model and state the clustering problem. Our clustering techniques (Chemical reaction optimization and game theory) are presented in section 3.5 and 3.6 respectively. Section 3.7 discusses their asymptotic analysis. Results are presented in Section 3.8, followed by the discussion in Section 3.9. Finally, Section 3.10 concludes this chapter.

3.3 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:

- Reducing the search space from an algorithmic perspective to help find faster solutions for the algorithms we have developed to deploy a service request.
- Build on-demand clusters such as CPU intensive clusters, bandwidth efficient clusters…

Figure 3.1 shows the VALKYRIE architecture we designed for the clustering of cloud/enterprise network topologies. It comprises a control plane and a data plane.

VALKYRIE builds clusters based on on-demand requirements, which could relate to the number of clusters we want to form, as well as their nature; i.e., we may want the clusters to be CPU-intensive, delay/bandwidth-efficient, single metric or multi-metric efficient clusters. Once the decision makers choose an option, 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 done according to the size of the underlying network topology (small-scale, medium-scale and large-scale networks).

Another important task worth mentioning is the preprocessing 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, we will not consider it during the clustering process. All these approaches and pre-processing tasks take place on the control

plane. 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).

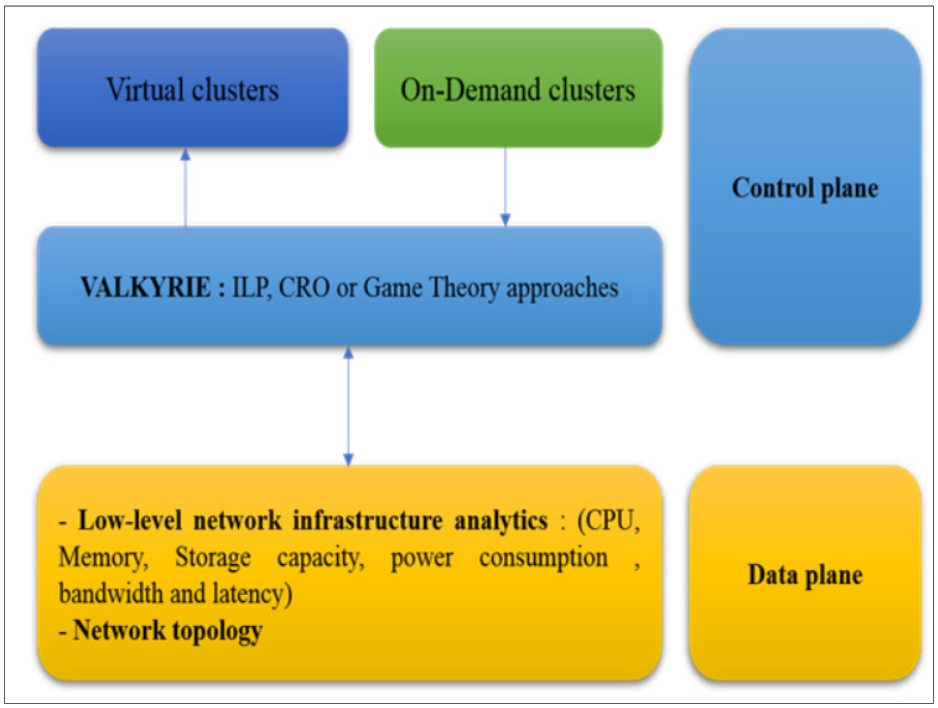


Figure 3.1 VALKYRIE for building virtual clusters for cloud and virtual network services

3.4 Related work

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.

Clustering in cloud environments:

Chen et al. (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 et al. (2015)), 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 (Sefraoui et al. (2012), Jackson et al. (2013)), 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 et al. (2014), propose a technique to cluster virtual machines in order to facilitate their placement 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.

Pongsakorn et al. (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 can communicate 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 higher bandwidth consumption rates are all but a given.

Mahmoud et al. (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 considers the application's architecture and characteristics. The resulting clusters (web cluster, database cluster, applications cluster, 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.

A.Wahab et al. (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 et al. (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.

Jami et al. (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, 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. (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 the "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 indifferent 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, 5G and network slicing.

3.5 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 meta-heuristic based approach.

3.5.1 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).

3.5.2 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 (Bothorel et al.2015) 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. 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.

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

- The structural quality of the clusters. We consider the modularity function (Clauset et al. (2004)), where a higher modularity corresponds to better clustering. Thus, here we need to maximize this measure as an objective function.
- 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.

3.5.3 VALKYRIE clustering model

Table 3.1 summarizes the different notations used throughout the paper.

Table 3.1 Notation table

C	Set of clusters.
Sim(i,j)	Entropy of a cluster Γ_c.
δ(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.
x_i^d	Vector of attributes associated to the server i.
$\gamma_i^c \in \{0,1\}$	γ_i^c= 1 if server i belongs to cluster c.

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.

- 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.

$$F1 = \frac{1}{2} \times \sum_{\forall c \in C, \forall i \in V, \forall j \in V} Sim(i,j) \tag{3.1}$$

Where,

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

- F2 tries to maximize the modularity for a better clustering (Clauset et al. (2004)).

$$F2 = \frac{1}{2|E|} \sum_{\forall i \in V, \forall j \in V, \forall c \in C} [A_{ij} - \frac{k_i k_j}{2|E|} \delta(i,j)] \tag{3.2}$$

Where, $A_{ij} = 1$ i,j if i and j are directly connected and otherwise, A_{ij} i,jis the adjacency matrix of the network and k_i the degree of server i. The degree refers to the number of connections each server has to its neighbors.

$\delta(i,j)$ 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 the decision variables which is equivalent to the Kronecker delta.
Thus, F2 becomes:

$$F2 = \frac{1}{2|E|} \sum_{\forall i \in V, \forall j \in V, \forall c \in C} [A_{ij} - \frac{k_i k_j}{2|E|} \gamma_i^c \gamma_j^c] \tag{3.3}$$

Constraints

$$\forall c \in C: \sum_{i \in V} \gamma_i^c = 1 \tag{3.4}$$

$$\forall\, i \in V: \sum_{c \in C} \gamma_i^c \geq 2 \tag{3.5}$$

$$\forall\, c \in C, \forall\, i \in V: \gamma_i^c \in 0,1 \tag{3.6}$$

Constraint (3.4) ensures that each server belongs to only one cluster. Thus, the model is outlier-free. Constraint (3.5) 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 (3.6) specifies that the decision variable is binary (its value is either 1 or 0).

3.6 CRO-based clustering approach

3.6.1 CRO approach description

Chemical reaction optimization is a population-based metaheuristic 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 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 et al. (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.

3.6.2 Molecule encoding

One possible way to encode the solution is to use the grouping technique as defined in (El Mensoum et al. (2019)) where each cluster will represent the group of servers it contains as described in Table 3.2.

Table 3.2 First Molecule encoding

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

Table 3.2 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 (3.3) 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 3.3.

Table 3.3 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.

3.6.3 Initial population

Our CRO approach consists of a centroid-based clustering where we specify a set of points in the dataset and then assign the rest of our servers to one of the defined centroids based on the similarity measure, we defined in section 3.5.2.

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 dataset 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 3.1.

Algorithm 3.1 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 != 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 dataset variance.

8: **end while**

3.6.4 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 5.1 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 3.2 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 3.2). 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.

Algorithm 3.2 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 new 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**

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. 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 3.3) and then assign the rest of nodes to their closest centroids based on similarity (step 3 Algorithm 3.3).

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 3.3) and assign servers 0 to R to their closest cluster again based on similarity (step 5 Algorithm 3.3). 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.

Algorithm 3.3 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**

- **Multi-molecular reactions:**

The first operator in this category is the Synthesis, and its full process is described in Algorithm 3.4. 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 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 2^{nd} parent ranging from server[R+1] to server[N] where is the number of servers considered in our dataset (step 2 in Algorithm 3.4). 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 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 3.4).

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 3.2 in order to generate two distinct solutions: ***C1=ON-WALLCOLLISION (M1) and C2=ON-WALL COLLISION (M2).***

Algorithm 3.4 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**

3.6.5 Over all CRO-based clustering Algorithm

In Algorithm 3.5 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 1. We start first by generating a random number **B** between [0,1] (step 4 Algorithm 3.5) if **B** is higher than Mol then we randomly select one Molecule from the initial population generated by Algorithm 3.1. 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 3.5), then we apply the decomposition Algorithm 3.3. Otherwise, the on-wall ineffective collision Algorithm 3.2 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 3.5), then the synthesis operator (Algorithm 3.4) is executed. Otherwise, we apply the inter-molecular ineffective collision process and we decrement 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 3.5.3. The solution that returns the highest modularity is selected as the optimal solution.

Algorithm 3.5 Over-all CRO algorithm for clustering

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**

1: Generate randomly a number **Mol** in the range [0,1]
2: Generate initial molecules by executing Algorithm 3.1
3: **while** Num-Hits <MaxIteration **do**
4: Generate randomly **B** in the range [0,1]
5: **If** B >Mol **do**
6: Randomly select one molecule
7: **If** Num-hits >A **do**
8: Execute Decomposition Algorithm 3.3
9: **else**
10: Execute On-wall ineffective collision Algorithm 3.2
11: **end If**
12: **else**
13: Randomly select two molecules from the population
14: **If** KE <B **do**
15: Execute Synthesis Algorithm 3.4
16: **else**
17: Execute Inter-molecular ineffective collision
18: **end If**
19: KineticEnergy–
20: **end If**
21: Num-Hits ++
22: **end while**

23: Check for a new optimal solution returning the highest modularity
24: **return** Final Solution **O** and its modularity

3.7 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 S_i and every other server S_j in the substrate network. The similarity values calculated were ranked in descending order and formed the preference table as shown in Algorithm 3.6.

Algorithm 3.6 Preference list computation

Input: Set of attributes for all servers ***SA***, list of servers ***S***

Output: Preference List

1. **for each** $S_i \in S$
2. **for each** $S_j \in S$
3. Calculate the similarity between servers' S_i and S_j and store them in sim array.
4. Sort sim array in descending order to form 2dimensional sim array.
5. **end for**
6. **end for**
7. **return** Preference List

After the preference table is built, each server should be uniquely paired with only one other server in the substrate, which is accomplished by applying the Irving Algorithm (R.W.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 3.6. The Irving algorithm matches each server to only one other server. Each server Si sends its proposal to the most preferable server Sj 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 3.2.

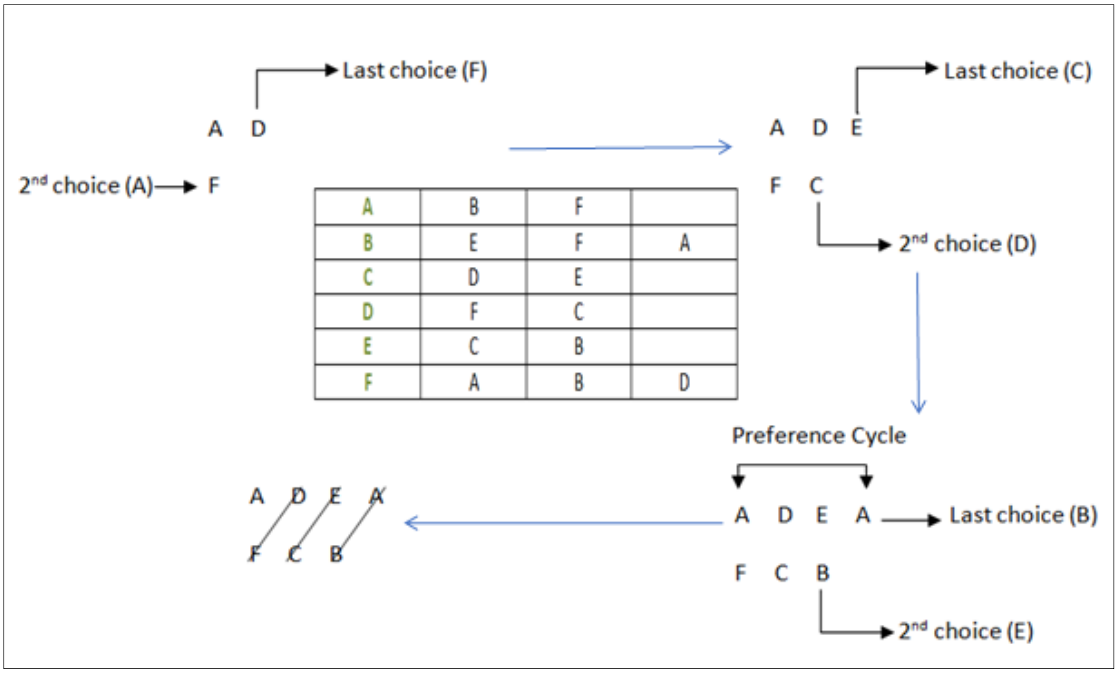


Figure 3.2 Preference cycle elimination process

The last step in the Irving algorithm is to eliminate all the preference cycles existing in the shortlisted preference table.

Algorithm 3.7 Irving Algorithm

Input: The servers set S and their attributes.
Input: The preference list.
Output: The stable matching.

1. Matching-pairs = Irving(*S*, *PreferenceList*)
2. **Return** matching pairs

This process is performed by checking all the rows in the shortlisted preference table, which has more than one preference. For instance, as shown in Figure 3.2, 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 3.2. 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 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 3.8.

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 3.8 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.

Algorithm 3.8 Clusters improvement and forming

Input: Matching pairs, Clusters

Output: Clusters of servers

1. **For each** Cluster in Clusters
2. **For each** pair in matching pairs
3. Check the similarity between each server in pair and the centroid of the cluster
4. **if** (Similarity exists) then
5. **if** (already in cluster) then
6. keep the server inside the cluster
7. **else**
8. **if** (adding this pair to the cluster maximizes modularity) then
9. join the cluster
10. remove the pair from the matching pairs
11. **else**
12. do not join the cluster
13. check the next pair from the matching pairs
14. **end if**
15. **end if**
16. **end if**
17. **end for**
18. **end for**

Return clusters of servers

3.8 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 embodies the form of the well-known Quadratic Assignment Problem (Sahni et al. (1976)).

CRO-based approach:

Our CRO-based approach consists of four algorithms, as described in section 3.6. We detail the complexity of each one as follows:

- Algorithm 3.1 runs in O(Popsize × V)
- Algorithm 3.2 runs in O(V^2+ V)
- Algorithm 3.3 also runs in O(V^2 + V)
- Algorithm 3.4 runs in O(V^2)

Based on this analysis, Algorithm 3.5 (CRO-based approach) complexity is

$$\mathbf{O(MaxIteration \times (V^2 + V))}$$

Game theory-based approach:

Similarly, three algorithms were employed and their complexities are computed as follows:

- Algorithm 3.6 runs in O (V^3×Log(V)) as we use TimSort algorithm to sort the similarity array which runs in O(V × Log(V)) (Cormen et al. (2009))
- Algorithm 3.7 runs in O(V^2)
- Algorithm 3.8 runs in O(V^3)

Based on this analysis, the Game Theory-based approach complexity is

$$\mathbf{O(V^2 + V^3 \times Log(V))}$$

3.9 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.

3.9.1 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.

3.9.2 Performance metrics

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

Runtime: we calculate the time taken by the different approaches to partition the network into the number of desired clusters.

Similarity: we evaluate the average similarity of all the clusters.

Modularity: we evaluate how dense the connections between the nodes are within the clusters and how sparse they are while in different clusters.

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.

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.

3.9.3 Scenarios

Our approaches are evaluated under a set of scenarios, which we define hereunder.

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.

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 3.4 Degrees of connectivity in our scenarios

Network size	scenario 1	scenario 2	scenario 3
20	2	5	3
50	7	7	3
100	14	13	3
200	29	25	3
300	44	35	3
500	74	75	3
1000	149	150	3

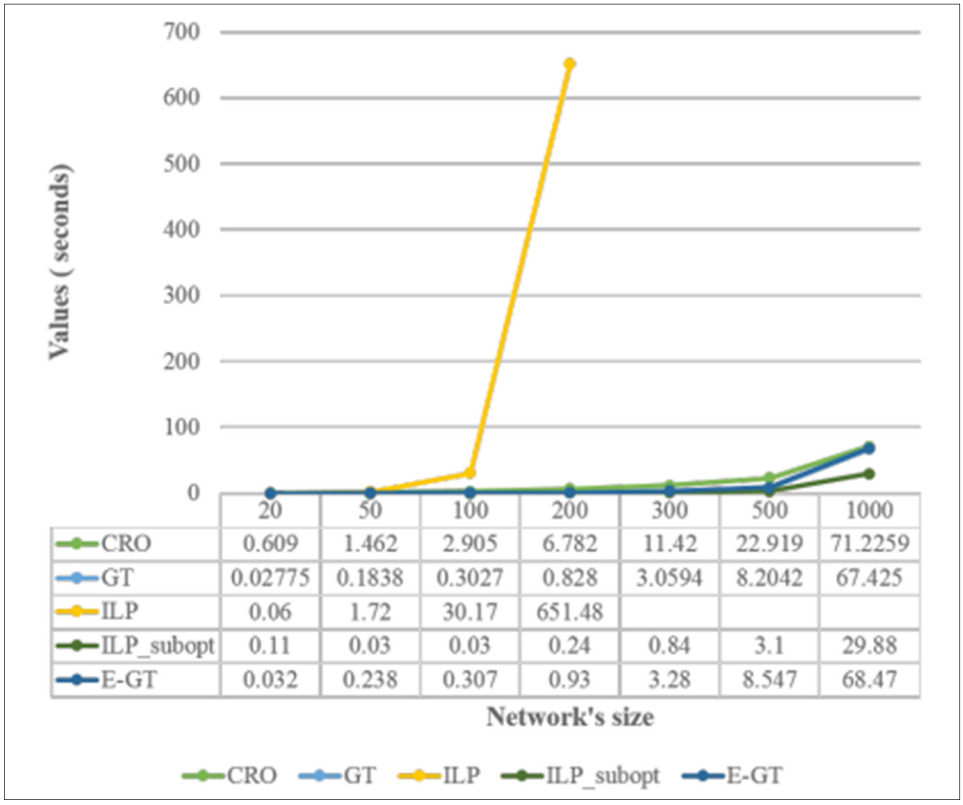
3.10 Discussion and observations:

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 to obtain quick solutions.

3.10.1 Modular topology with variable connectivity degree

Based on Figure 3.3, 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 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 67 seconds for GT and from 0.03 to 68 seconds for EGT, 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.



	20	50	100	200	300	500	1000
CRO	0.609	1.462	2.905	6.782	11.42	22.919	71.2259
GT	0.02775	0.1838	0.3027	0.828	3.0594	8.2042	67.425
ILP	0.06	1.72	30.17	651.48			
ILP_subopt	0.11	0.03	0.03	0.24	0.84	3.1	29.88
E-GT	0.032	0.238	0.307	0.93	3.28	8.547	68.47

Figure 3.3 Clustering time for modular topology

Another metric that requires evaluation is the similarity, as shown in Figure 3.4, 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.

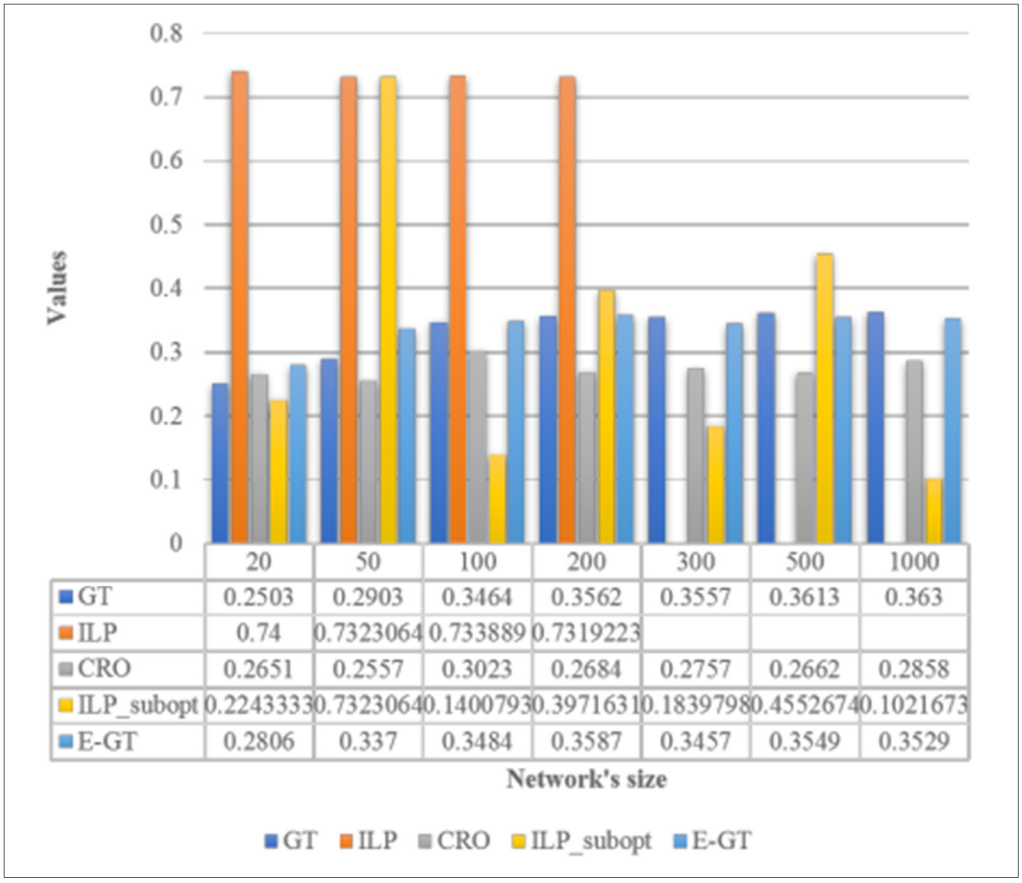


Figure 3.4 Similarity values for modular topology

With regard to the Game Theory approach, not all servers are included in the formed clusters, which explains the difference in similarity, albeit that this difference is fairly small at an average of 0.05. Figure 3.5 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 3.4, ILP finds the best similarity but Figure 3.5 illustrates that this has an impact on the modularity.

	20	50	100	200	300	500	1000
CRO	0.0528	0.0216	0.01731	0.01431	0.01478	0.014045	0.022485
GT	0.1348	0.0575	0.0045	0.0157	0.0579	0.0195	0.0253
ILP	-0.041551	-0.001516	-0.002022	-4.43E-05			
ILP_subopt	0.0124654	-0.001516	-0.002294	0.0001133	-0.000251	-9.65E-05	1.22E-06
E-GT	-0.0128	-0.01604	-0.01491	-0.0041	-0.00364	-0.000872	-0.00142

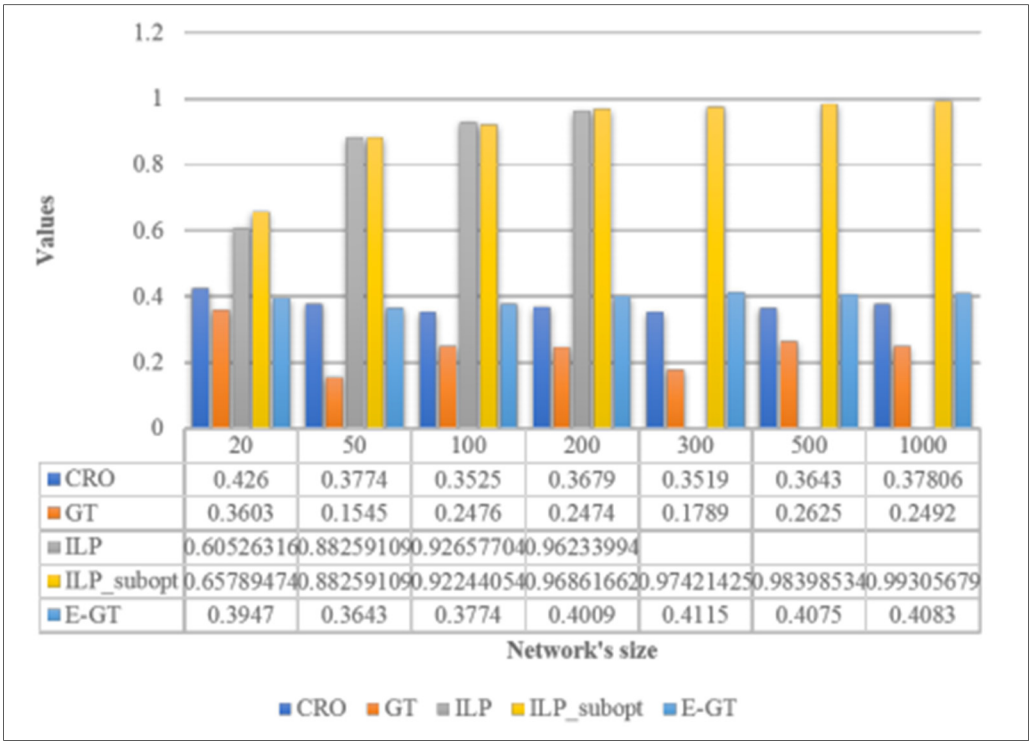
Figure 3.5 Modularity values for modular topology

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. As shown in Table 3.5 that the number of outliers increases in a linear manner as the network size increases. This only occurred in the GT approach, as none of the other approaches has outliers and all of them are respecting all the constraints defined in the mathematical model. Some outliers are sacrificed in order to achieve better values for similarity and modularity, 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.

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 3.6, 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 denser 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 exhibit the highest density, it is not deployable since some clusters contain only two servers, which is not a preferable cluster for the network administrator to have.

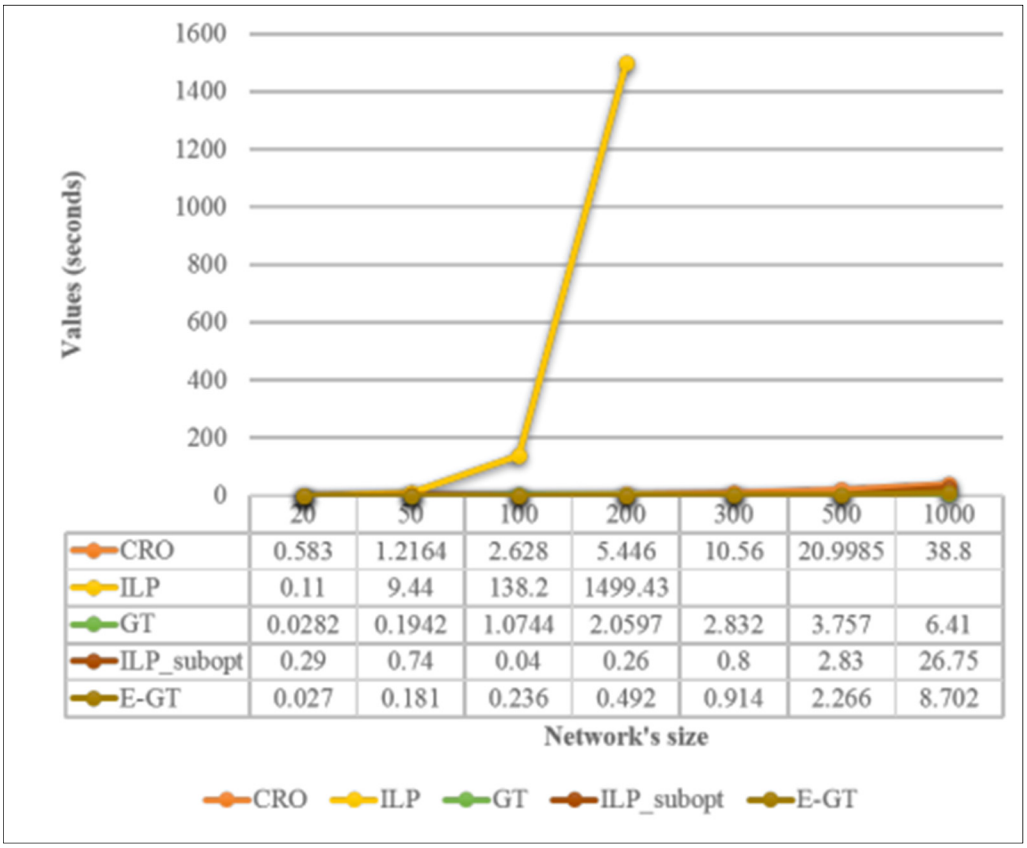


	20	50	100	200	300	500	1000
CRO	0.426	0.3774	0.3525	0.3679	0.3519	0.3643	0.37806
GT	0.3603	0.1545	0.2476	0.2474	0.1789	0.2625	0.2492
ILP	0.60526316	0.88259109	0.92657704	0.96233994			
ILP_subopt	0.65789474	0.88259109	0.92244054	0.96861662	0.97421425	0.98398534	0.99305679
E-GT	0.3947	0.3643	0.3774	0.4009	0.4115	0.4075	0.4083

Figure 3.6 Density values for modular topology

3.10.2 Random topology with fixed connectivity degree

Like the previously tested topologies, and as shown in Figure 3.7 the ILP solution takes a considerable amount of time to converge to optimal 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.



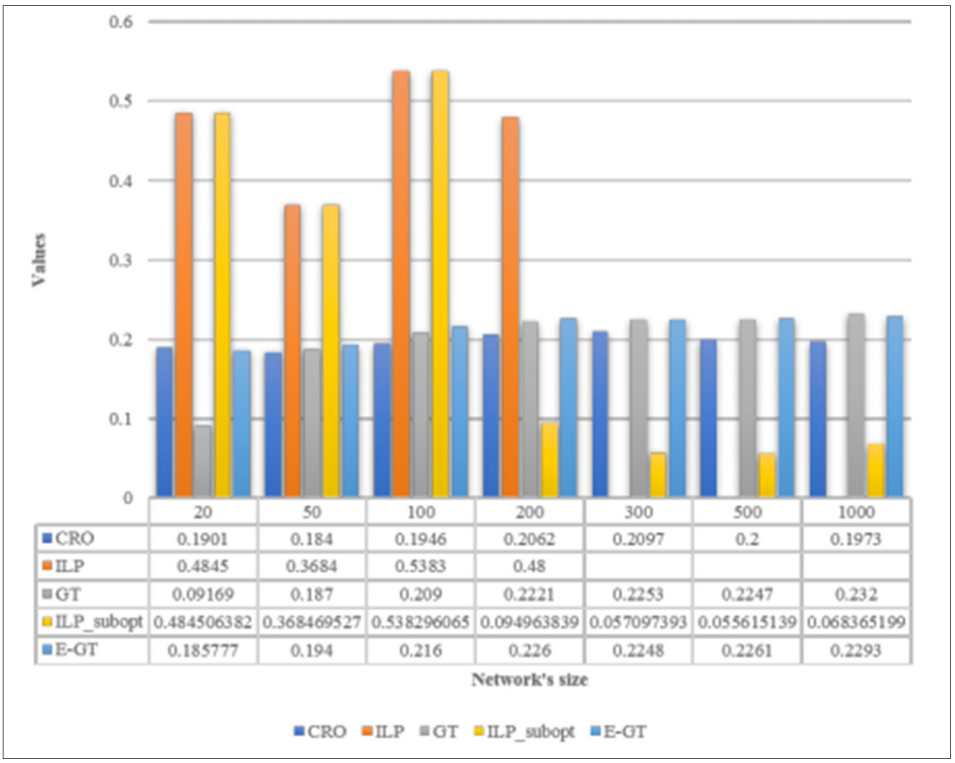
	20	50	100	200	300	500	1000
CRO	0.583	1.2164	2.628	5.446	10.56	20.9985	38.8
ILP	0.11	9.44	138.2	1499.43			
GT	0.0282	0.1942	1.0744	2.0597	2.832	3.757	6.41
ILP_subopt	0.29	0.74	0.04	0.26	0.8	2.83	26.75
E-GT	0.027	0.181	0.236	0.492	0.914	2.266	8.702

Figure 3.7 Clustering time for random topology with fixed connectivity degree

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. There is not a big difference for 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 result compared to CRO but again with an existence of outliers which explains this difference.

In Figure 3.8, 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. There is no major difference between the behaviors of the CRO, GT and E-GT approaches in terms of similarity when the degree is fixed because this measure is more related to the servers' attributes than to the connectivity between them. GT finds slightly better results than does CRO, but again, with the existence of outliers, which explains this difference.

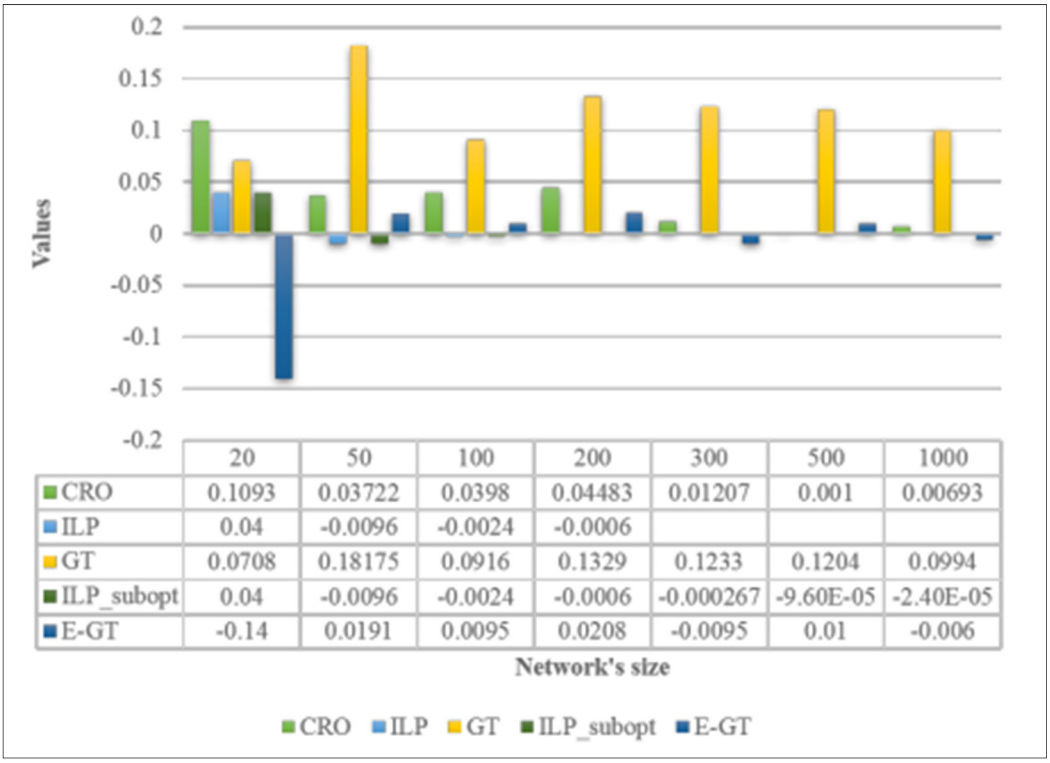


	20	50	100	200	300	500	1000
CRO	0.1901	0.184	0.1946	0.2062	0.2097	0.2	0.1973
ILP	0.4845	0.3684	0.5383	0.48			
GT	0.09169	0.187	0.209	0.2221	0.2253	0.2247	0.232
ILP_subopt	0.484506382	0.368469527	0.538296065	0.094963839	0.057097393	0.055615139	0.068365199
E-GT	0.185777	0.194	0.216	0.226	0.2248	0.2261	0.2293

Figure 3.8 Similarity values for random topology with fixed connectivity degree

The difference of modularity is higher in this topology between GT and all the other approaches as shown in Figure 3.9. 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 topologies, it still finds poor values for modularity but the best results for similarity.



	20	50	100	200	300	500	1000
CRO	0.1093	0.03722	0.0398	0.04483	0.01207	0.001	0.00693
ILP	0.04	-0.0096	-0.0024	-0.0006			
GT	0.0708	0.18175	0.0916	0.1329	0.1233	0.1204	0.0994
ILP_subopt	0.04	-0.0096	-0.0024	-0.0006	-0.000267	-9.60E-05	-2.40E-05
E-GT	-0.14	0.0191	0.0095	0.0208	-0.0095	0.01	-0.006

Figure 3.9 Modularity values for random topology with fixed connectivity degree

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

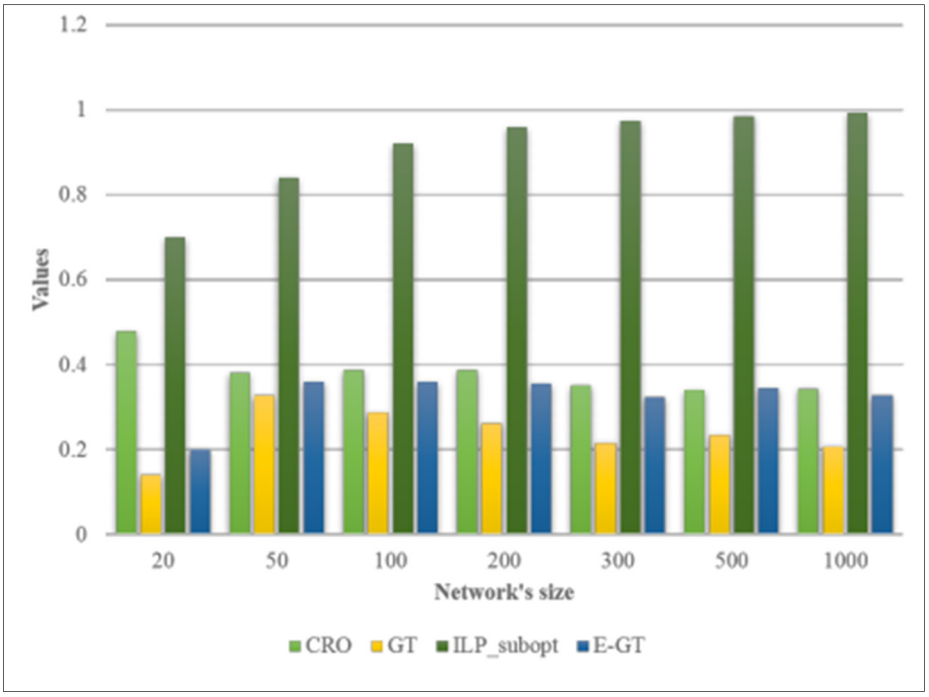


Figure 3.10 Density values for Random topology with fixed connectivity degree

In Table 3.5, the number of outliers is increasing linearly as the network size is increasing only in GT approach, all other approaches do not have outliers and respecting all the constraints defined in the mathematical model.

We sacrifice having some outliers in order to have a better value for similarity and modularity. This could be beneficial for cloud provider since there might be some use cases where SFC request has a specific requirements and preference to be deployed a cluster with higher modularity and similarity values.

Table 3.5 Number of outliers in modular topology

Approach	**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

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 (≤ 200 servers), whereas GT has 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.

3.11 Conclusion

In this chapter, 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 tradeoff between density, modularity, similarity and cost functions, in addition to being outliers-free. One of the solutions may 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, 5G and network slicing. Finally, given their low computational complexities, our solutions could be integrated into orchestration systems following the NFV MANO framework.

CHAPTER 4

DISCUSSION OF THE RESULTS

In this thesis, we proposed two main approaches for service functions chains placement and chaining for clustering of substrate networks which are intended to improve the management of resources in the NFV platforms.

The first proposed tool for placement and chaining of SFCs consists of a multi-stage approach to ensure an optimal embedding technique minimizing the provider's costs whether in term of operational, transmission or penalty costs. To achieve this goal, we used a set of metaheuristics mainly the chemical reaction optimization algorithm as well as the genetic algorithm. The experimental analysis shows that our proposed approach is efficient for VNF placement and chaining as it allows taking many metrics into account in the optimization process (CPU, memory, energy, delay…). The results also show that the proposed algorithms scale well with large network architectures and take also into account the dynamic aspect of resources in cloud-based environments where the available resources are constantly changing.

However, the proposed approach can still be improved by considering also the dependencies between VNFs when making the decision of placement and chaining. For example, if two VNFs belonging to the same SFC require a lot of CPU of memory they should probably not be placed on the same server, because in high load traffic they might result in some bottlenecks over that hosting server. So, one way to improve the proposed approaches is to consider also the relationships between VNFs prior to placing and chaining them. Moreover, one of the most used orchestration tools actually is Kubernetes and the way it places containers/VNFs does not consider this aspect of dependencies between VNFs which leaves the door wide open to work on this aspect in the research field.

Our second approach proposes a CRO-based clustering technique that allows dividing the substrate network into a set of on demand clusters or group of servers. This pro-active measure helps to prepare a set of efficient servers in terms of selected attributes like CPU, memory or energy. The solution helps also to reduce the search space for other operation like placement, scaling or migration of VNF components. The experimental tests that we conducted on different network topologies prove that the quality of the formed clusters is good when using metaheuristic such as those based on CRO and Game-theory techniques. The results also give the cloud provider the choice to use one of those techniques according to the optimization goal and also depending on the nature of the networking set on the physical substrate.

The techniques proposed in our second contribution help improving the quality of the formed clusters. In fact we consider multi-attribute clusters, contrary to what we proposed in the first article using k-medoids clustering, where we only considered one attribute at the time and each formed cluster is efficient only in terms of that single attribute. In some cases, the cloud provider would like to have cluster that are both efficient in terms of CPU and memory for example, so our second proposed approach is tailored to serve these use cases also.
Apart from the multi-dimension aspect, our second approach based on CRO and Game-theory algorithms leverages also the connectivity between servers which was not considered in the k-medoids clustering. The nature of the set networking on the substrate infrastructure has a major impact on the quality of the formed clusters, because it ensures that servers belonging to the same clusters are connected enough between them to avoid resource wastages in terms of bandwidth and guarantee low latencies for the hosted SFCs.

CONCLUSION and RECOMMANDATIONS

In the present thesis, we proposed two main approaches that are complementary for the placement and chaining of virtual network functions in cloud environments. Both solutions resulted in one journal paper accepted for publication, one submitted journal paper and one provisional application patent for the clustering technique.

In Chapter 2, we described the first proposal which uses a clustering-based approach to reduce the search space and perform the most optimal placement and chaining of virtual network functions over shared physical infrastructures. The experimental analysis showed that our proposed algorithms are efficient to consider multi-objective optimization problems as we managed to perform the placement in short execution time and we also proved that our approaches ensure a good optimization level in terms of the physical resources in the substrate network topologies as well as better latencies and delay values. We didn't make any comparisons to the already proposed solutions in the literature, as to the best of our knowledge, there was no CRO-based placement and chaining approaches proposed in the previous research works.

In Chapter 3, we present our second contribution that consists in clustering the substrate network. This process helps to prepare the network and divide it into a set of efficient groups in terms of selected attributes that could be either memory, CPU or Energy. Once these homogenous groups of servers are formed, they guarantee that the requested resources by the incoming SFC requests are available and ensure that the placement will be performed in a fast manner. During the experimental phase, we stressed our proposed techniques with different network topology sizes, with servers having different attributes. Results show that the adopted algorithms scale very well with large topologies and return well partitioned clusters. Moreover, as we propose different solutions based on different meta-heuristic algorithms, our paper gives cloud providers a large choice to decide which of the techniques to adopt based on their targeted optimization metrics and depending on the nature of the physical network and its networking.

The combination of these two proposals ensure better resource management and enhance the functionalities of the NFV MANO platform, as the clustering pre-process the network topology which reduces the search space and guarantees the availability of resources on the chosen cluster. The placement and chaining are then performed on the processed network to improve the quality of service deployment and provisioning in cloud-based environments.

As perspectives for future work, we intend to extend our evaluations for the placement and chaining to consider other types of SFCs, in the first paper we tested our proposed approach on linear SFCs only, so it is interesting also to evaluate the behavior of our metaheuristics with other types of SFCs (non-linear, with replicas, etc.). It is also important to analyze the inter-dependencies and relationships between VNFs of the same SFC which has a major impact on the placement decisions, VNFs with high resource-consumption rates should not be placed on the same server, to avoid having bottlenecks when the traffic loads get higher.
Another enhancement to our proposed placement and chaining solution, could be to include other objective functions and consider other metrics in the optimization process like bandwidth, reliability or availability, etc.

For the clustering approach, as we didn't have any inputs from the industry on how data centers are designed and how the networking is set on them, we run our proposed approaches on random topologies but if we manage to get significant descriptions of a real cloud data center we would like to re-run the tests to evaluate the accuracy of our proposed solutions on a real case scenario. As future improvements, we also consider testing an approach where the number of clusters is dynamically updated depending on the state of resources on the substrate network. We also plan to explore other metaheuristic algorithms to solve the formulated problem. From the experimental results we concluded that each of the proposed approaches (CRO or Game theory based) performs well in terms of either similarity or modularity, but it is interesting to test other metaheuristics that may find better tradeoffs between these two objective functions and of course in reasonable execution time.

APPENDIX A

K-MEDOIDS CLUSTERING PROCESS: CLUSTERHEAD SELECTION AND CLUSTER FORMATION

A.1 K-medoids Clustering process

A.1.1 Initial Cluster-heads Selection

In the following, we present the proposed method to optimize the selection of the initial set of cluster-heads which is a statistical technique. The main idea is to compute the variance between each server in the substrate network and all the remaining servers, in terms of a set of metrics that we aim at optimizing (e.g., energy, bandwidth), the 2nd step consists of computing the variance of the whole set of servers. Based on these values, the servers whose variance (with respect to the other servers in the network) is less than the whole servers' set variance are pre-selected as eligible candidates to act as cluster heads. The main idea of this technique is to exclude the outliers (e.g., in terms of energy) that are too far from the central region, which have high variance values. Once the set of servers that are eligible to act as cluster-heads is obtained, we can filter the defined set of candidates even more. To achieve this, we compute the (Euclidean) distance between each pair of servers by respect of some pre-set attributes and choose the server which minimizes the distance to all other servers in the substrate network as the first cluster-head. To ensure the coherence of our clustering technique and avoid forming overlapping clusters, the server that maximizes the distance from the first cluster-head is chosen as the second cluster-head. Having selected two initial cluster-heads, we then need to increase the number of cluster heads up to k (the value of k is specified by the network administrators). This is achieved through a continuous search process (for k times) for the servers that maximize the distance from the previously selected cluster heads. This process, along with the mathematical formulas, is explained in what follows:

Step 1: Compute the Euclidean distance d(si;sj) between each pair (si;sj) of servers based to a defined set of K attributes using Eq. (A A-1).

$$d(s_i, s_j) = \sqrt{\sum_{k=1}^{K} (s_i^k - s_j^k)^2} \quad \text{(A A-1)}$$

Step 2: Compute the variance of all the servers, i.e., the variance among all servers and their mean using Eq. (A A-2).

$$\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} d(s_i - \bar{s})^2} \quad \text{(A A-2)}$$

$$\bar{s} = \sum_{i=1}^{n} \frac{s_i}{n}$$

Step 3: Compute the variance between each server si and all other servers using Eq.(A A-3)

$$\sigma_i = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} d(s_i, s_j)^2} \quad \text{(A A-3)}$$

Step 4: Determine the set P of cluster-head candidates as the ones having variance that is less (or equal) than the whole server set's variance proportionally to a stretch factor ω as depicted in Eq. (A A-4)

$$P = \{s_i | \sigma_i \leq \omega\sigma, i = 1, \dots, n\} \quad \text{(A A-4)}$$

Step 5: Compute the distance di of each server s_i (from all other servers) using Eq(A A-5):

$$d_i = \sum_{j=1}^{n} d(s_i, s_j) \quad \text{(A A-5)}$$

Step 6: Select the first cluster-head p_1 as the one that minimizes the distance with all other servers using Eq. (A A-6):

$$p_1 = argmin\,\{d_i | i = 1, .. n\} \quad \text{(A A-6)}$$

Step 7: Select the second cluster-head p_2 in such a way to maximize the distance with the previously selected cluster head p1 as depicted in Eq. (A A-7):

$$p_2 = argmin_{s_{i \in P}}\,\{d(s_i, p_1) | i = 1, .. n\} \quad \text{(A A-7)}$$

Step 8: Select k new cluster-heads within each cluster as the ones that are the farthest from their current medoid (p_1 or p_2) using Eq. (A A-8):

$$p' = argmin_{s_{i \in c \in P}}\,\{d(s_i, p_i) | i = 1, .. n\} \quad \text{(A A-8)}$$

Once the first initial set of k cluster-heads is elected, the following step consists of optimizing the selection process to ensure an improved clustering performance and quality. In the following section, we present the complete clustering algorithm and discuss its details.

A.1.2 Cluster Formation Algorithm

The stepwise process to execute the clustering is described in Algorithm A-1. Defining the set of initial cluster-heads is described is Steps (7) to (13) which implement the statistical technique we described in the previous section. Once the initial cluster-heads are defined, the following steps are repeated until no change occurs in the selected cluster-heads. The remaining servers are then assigned to the closest cluster-head (in terms of the considered attributes) (Step 15). Then, we compute in step 16 the clustering cost D1 (i.e., sum of distances from each cluster-head to its cluster members) is computed as per Eq. (A A-9)

$$\sum_{k=1}^{K} \sum_{z \in C_k} d(p_k, z)^2 \qquad \text{(A A-9)}$$

After that, for each cluster, we select the server which minimizes the distance from the other servers in the same cluster as are placement cluster-head (Steps 17-18). Consequently, each server is assigned again to the closest newly appointed cluster-head (Step 19) and the new clustering cost D2 is computed. In case the new clustering cost D2 is equal to the previous one D1 (Step 21), the algorithm stops and the clustering topology remains the same. Otherwise, Steps 15-21 are repeated until reaching a stable clustering topology.

Algorithm A-1 K-medoids substrate network clustering

1: **Input:** Set of physical servers
2: **Input:** Number K of clusters
3: **Input:** Set II = {energy, bandwidth, CPU, delay} of metrics to be minimized.

4: **Output**: Set $C_n = \{C_1,..,C_k)$ of clusters
5:**Output:** Set H = $\{h_1,...\ ,h_k\}$ of cluster-heads such that each cluster-head $h \in H$ is efficient in terms of metrics $\theta \in \prod$.

6: **procedure** CLUSTERING
7: Compute the distance between each pair of servers using Eq. (A.A-1)
8: Compute the variance of the whole set of servers using Eq. (A.A-2)
9: Compute each server's variance with regards to all other servers using Eq. (A.A-3)

10: Define the set of candidate cluster-heads using Eq. (A.A-4)
11: Choose two initial cluster-heads H = $\{h_1, h_2\}$ using Eqs. (A.A-6) and (A.A-7)
12: Allocate each server to the closest cluster-head and compute the total clustering cost using Eq. (A.A-9).
13: Increase the number of clusters up to k using Eq. (A.A-8).
14: Repeat

15: Assign each server to the closest cluster-head
16: Compute the sum $D_1$of distances from all cluster members to their cluster-head using Eq. (A.A-9).
17: Select a new cluster-head of each cluster as the one that minimizes the total distance from the other servers in its cluster.
18: Update the current cluster-head in each cluster by substituting it with the newly chosen cluster-head.
19: Assign each server to the closest cluster-head.
20: Compute the sum $D_2$of distances from all cluster members to their cluster-head using Eq. (A.A-9).

21: **If** $D_2 = D_1$, stop the algorithm; otherwise go back to step 16.
22: **end**

23: **end** procedure

BIBLIOGRAPHY

B. Yi, X. Wang, K. Li, M. Huang et al., "A comprehensive survey of network function virtualization," Computer Networks, 2018.

J. Gil-Herrera and J. F. Botero, "Resource allocation in nfv: A comprehensive survey," IEEE Transactions on Network and Service Management, vol. 13, pp. 518–532, 2016.

M. Xia, M. Shirazipour, Y. Zhang, H. Green, and A. Takacs, "Network function placement for nfv chaining in packet/optical datacenters," Journal of Lightwave Technology, vol. 33, no. 8, pp. 1565–1570, 2015.

S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on. IEEE, 2014, pp. 7–13.

M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba, "Elastic virtual network function placement," in Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on. IEEE, 2015, pp. 255–260.

X. Sun and N. Ansari, "Primal: Profit maximization avatar placement for mobile edge computing," in Communications (ICC), 2016 IEEE International Conference on. IEEE, 2016, pp. 1–6.

A. Alleg, T. Ahmed, M. Mosbah, R. Riggio, and R. Boutaba, "Delay aware vnf placement and chaining based on a flexible resource allocation approach," in Network and Service Management (CNSM), 2017 13th International Conference on. IEEE, 2017, pp. 1–7.

S. Khebbache, M. Hadji, and D. Zeghlache, "A multi-objective non dominated sorting genetic algorithm for vnf chains placement," in Consumer Communications & Networking Conference (CCNC), 2018 15th IEEE Annual. IEEE, 2018, pp. 1–4.

V. Eramo, E. Miucci, M. Ammar, and F. G. Lavacca, "An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures," IEEE/ACM Transactions on Networking, vol. 25, no. 4, pp. 2008–2025, 2017.

R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in Computer Communications (INFOCOM), 2015 IEEE Conference on. IEEE, 2015, pp. 1346–1354.

C. Pham, N. H. Tran, S. Ren, W. Saad, and C. S. Hong, "Traffic-aware and energy-efficient vnf placement for service chaining: Joint sampling and matching approach," IEEE Transactions on Services Computing,2017.

J. Liu, W. Lu, F. Zhou, P. Lu, and Z. Zhu, “On dynamic service function chain deployment and readjustment,” IEEE Transactions on Network and Service Management, vol. 14, no. 3, pp. 543–553, 2017.

S. Draxler, H. Karl, and Z. ¨ A. Mann, “Jasper: Joint optimization of ´scaling, placement, and routing of virtual network services,” IEEE Transactions on Network and Service Management, 2018.

S. Lange, A. Grigorjew, T. Zinner, P. Tran-Gia, and M. Jarschel, “A multi objective heuristic for the optimization of virtual network function chain placement,” in Teletraffic Congress (ITC 29), 2017 29th International,vol. 1. IEEE, 2017, pp. 152 160.

J. Xu, A. Y. Lam, and V. O. Li, “Chemical reaction optimization for task scheduling in grid computing,” IEEE Transactions on Parallel and Distributed Systems, vol. 22, no. 10, pp. 1624–1631, 2011.

A. Y. S. Lam and V. O. K. Li, “Chemical reaction optimization for cognitive radio spectrum allocation,” in IEEE Global Telecommunications Conference GLOBECOM. IEEE, 2010.

N. Dezhabad and S. Sharifian, “Learning-based dynamic scalable load balanced firewall as a service in network function-virtualized cloud computing environments,” The Journal of Supercomputing, pp. 1–30,2018.

L. Yala, P. A. Frangoudis, and A. Ksentini, “Latency and availability driven VNF placement in a MEC-NFV environment,” in GLOBECOM2018, IEEE Global Communications Conference 9-13 December 2018,Abu Dhabi, UAE, Abu Dhabi, UNITED ARAB EMIRATES, 12 2018.

G. L. M. G. Yu, Donghua and X. Liu, “An improved k-medoids algorithm based on step increasing and optimizing medoids,” Expert Systems with Applications, vol. 92, pp. 464–473, 2018.

L. Lin and M. Gen, “Priority-based genetic algorithm for shortest path routing problem in ospf,” in Intelligent and Evolutionary Systems.Springer, 2009, pp. 91–103.

E. Falkenauer, “A hybrid grouping genetic algorithm for bin packing,”Journal of heuristics, vol. 2, no. 1, pp. 5–30, 1996.

C. Bothorel et al. Clustering attributed graphs: models, measuresand methods. Network Science, 3(03): 408-444, 2015.

Alessandro. B et al. Efficiently Clustering Very Large Attributed Graphs. arXiv:1703.08590v2.

Yang. Z et al. Graph Clustering based on Structural/Attribute Similarities. Proceedings of the VLDB Endowment 2.1 (2009): 718-729.

Hong. C et al. Clustering Large Attributed Graphs: A Balance between Structural and Attributes Similarities. ACM Transactions on Knowledge Discovery from Data (TKDD) 5.2 (2011): 12.

A. Clauset, M. E. Newman, and C. Moore. Finding community structure in very large networks. Physical review E 70.6 (2004):066111.

C. Chen, Y. Du, S. Chen and W. Wang, ”Partitioning and Placing Virtual Machine Clusters on Cloud Environment,” 2018 1st International Cognitive Cities Conference (IC3), Okinawa, 2018, pp. 268-270.

S.-J. Chen , C.-C. Chen , H.-L. Lu and W.-J. Wang, ”Efficient Resource Provisioning for Virtual Clusters on the Cloud,” in 2015International Conference on Platform Technology and Service, Jeju,Korea, 2015.

O. Sefraoui, M. Aissaoui, and M. Eleuldj, ”OpenStack: Toward an Open-Source Solution for Cloud Computing,” International Journal of Computer Applications, vol. 55, pp. 38-42, 2012.

K. Jackson and C. Bunch, OpenStack Cloud Computing Cookbook- Second Edition, 2 edition ed. Birmingham, UK: Packt Publishing,2013.

V. Chavan and P. R. Kaveri, ”Clustered virtual machines for higher availability of resources with improved scalability in cloud computing,” 2014 First International Conference on Networks &Soft Computing (ICNSC2014), Guntur, 2014, pp. 221-225.

P. U-chupala, P. Uthayopas, K. Ichikawa, S. Date and H. Abe, ”An implementation of a multi-site virtual cluster cloud,” The 2013 10thInternational Joint Conference on Computer Science and Software Engineering (JCSSE), Maha Sarakham, 2013, pp. 155-159.

M. Abdelsalam, R. Krishnan and R. Sandhu, ”Clustering-Based IaaS Cloud Monitoring,” 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), Honolulu, CA, 2017, pp. 672-679.

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.

S. Liu and Z. Li, ”A modified genetic algorithm for community detection in complex networks,” 2017 International Conference on Algorithms, Methodology, Models and Applications in Emerging Technologies (ICAMMAET), Chennai, 2017, pp. 1-3.

V. Jami and G. R. M. Reddy, ”A hybrid community detection based on evolutionary algorithms in social networks,” 2016 IEEE Students’ Conference on Electrical, Electronics and Computer Science(SCEECS), Bhopal, 2016, pp. 1-6.

Zachary, W. W., An information flow model for conflict and fission in small groups”, Journal of Anthropological Research 33 (1977),452-473.

A. Aylani and N. Goyal, ”Community detection in social network based on user as social activities,” 2017 International Conference onI-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC),Palladam, 2017, pp. 625-628.

Lam, A. Y., & Li, V. O. (2012). Chemical reaction optimization: Atutorial. Memetic Computing, 4(1), 3-17.

El Mensoum, I., Abdul Wahab, O., Kara, N., & Edstrom, C.(2019). MuSC: A Multi-Stage Service Chains Embedding Approach. Submitted to Journal of Network and Computer Applications.

Lam, A. Y., & Li, V. O. (2012). Chemical reaction optimization: Atutorial. Memetic Computing, 4(1), 3-17.

R. W. Irving, “An efficient algorithm for the ‘stable roommates ‘problem,” J. Algorithms, vol. 6, no. 4, pp. 577–595, 1985.

Sahni, Sartaj, and Teofilo Gonzalez. ”P-complete approximation problems.” Journal of the ACM (JACM) 23.3 (1976): 555-565.

J, P., Hong, P., Xue, K., & Li, D. (2018). Efficiently Embedding Service Function Chains with Dynamic Virtual Network Function Placement in Geo-distributed Cloud System. *IEEE Transactions on Parallel and Distributed Systems*, 1-1. doi: 10.1109/TPDS.2018.2880992.

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

ETSI GS NFV. (2013). Network Functions Virtualization (NFV) Architectural Framework. Norm ETSI 002 v1.1.1 (2013-10).