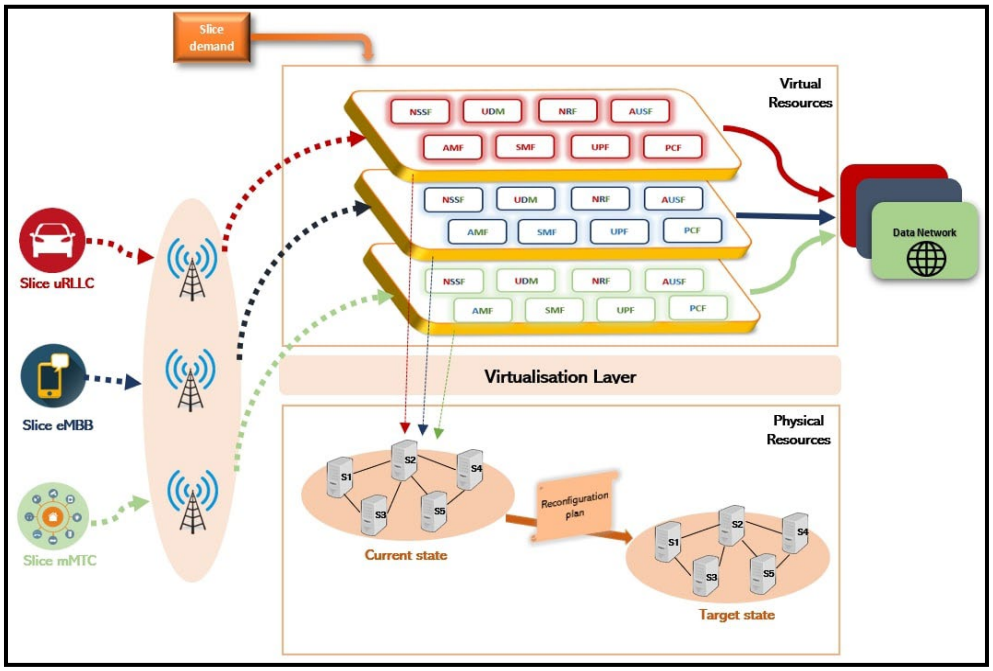
Optimization of VNF reconfiguration problem for 5G network slicing

Hanane Biallach

Par **Hanane BIALLACH**

Optimization of VNF reconfiguration problem for 5G network slicing

Thèse présentée
pour l'obtention du grade
de Docteur de l'UTC



Soutenue le 11 octobre 2022
Spécialité : Informatique : Unité de recherche Heudyasic (UMR-7253)

D2707

THESIS

presented by : **Hanane BIALLACH**
defended on : **October 11, 2022**

In order to become : **Doctor from UTC University**
prepared on **Compiègne University of Technology & Orange Innovation**

Spécialité : Informatique

Speciality : Computer Science

Optimization of VNF Reconfiguration Problem for 5G Network Slicing

Thesis supervised by :
Dritan NACE, Professor, UTC

and Co-Supervised by :
Mustapha Bouhtou, Dr Engineer, Orange

Committee members

Ghada JABER	Compiegne University of Technology	Examiner
Renaud SIRDEY	CEA List	Examiner
Hacene FOUCHAL	University of Reims Champagne-Ardenne	Referee
Soufian BENAMOR	University of Versailles Saint-Quentin	Referee
Dritan NACE	Compiegne University of Technology	Supervisor
Mustapha BOUHTOU	Orange Innovation	Co-Supervisor

To my beloved father's soul.

Acknowledgement

First and foremost, I must acknowledge and thank The Almighty Allah for giving me health, strength, and patience to conduct this work.

I would like to express my sincerest gratitude to all those who have supported and helped me throughout this period. In a first place, I would like to thank my supervisors Prof. Dritan Nace and Dr. Mustapha Bouhtou for their useful comments, remarks and engagement through the learning process of this PhD thesis. All the work was completed under their careful guidance and care. They devoted a lot of effort and energy and helped me to grow not only from a professional perspective but also as a person. Here, I would like to extend my high respect and heartfelt thanks to them.

Next, I would like to thank Prof. Hacene Fouchal and Prof. Soufian Benamor for the time and effort invested in the review and evaluation of this manuscript. I would also like to thank Dr. Renaud Sirdey and Mc. Ghada Jaber for being part of my evaluation committee as examiners.

I express my gratitude to Dr. Bertrand Decocq, head of the BRAINS team at Orange Innovation, for receiving me so warmly from my first day and providing me with all the support needed. I would also like to thank all the members of BRAINS team for making my integration in the team as easy, and for the incredible time and the great discussions. I'm deeply grateful for being part of the team !

Special thanks to my husband who provided me with moral support and bared my stress during the whole period. He always encouraged and motivated me to never quit until I get everything I dreamed of. Many thanks to my friends too who always stand by my side.

Finally, I would like to thank my wonderful mother, brother, and sister for their unconditional love and endless support. I really cannot thank them enough, they are the greatest blessing of my life. Without them, I would not have come this far.

Résumé

Ces dernières années, en raison de la croissance sans précédent du nombre d'appareils connectés et de données mobiles, et des développements continus des technologies pour répondre à cette énorme demande de données, le réseau de cinquième génération (5G) a émergé. La future architecture 5G sera essentiellement basée sur le Network Slicing (NS), qui permet de fournir une approche flexible pour réaliser la vision 5G. Grâce au concept émergent de virtualisation des fonctions réseau (NFV), les fonctions réseau sont découplées des matériels physiques dédiés et réalisées sous forme de logiciel. Cela offre plus de flexibilité et d'agilité dans les opérations commerciales.

Malgré les avantages qu'il apporte, NFV soulève quelques défis techniques, le problème de reconfiguration étant l'un d'entre eux. Ce problème, qui est NP-difficile, consiste à réaffecter les fonctions de réseau virtuel (VNFs) pour s'adapter aux changements du réseau, en transformant l'état courant des services déployés, on peut illustrer cela par la migration des machines virtuelles (VM) qui hébergent les VNF, à un autre état qui répond aux objectives des opérateurs.

Cette thèse de doctorat étudie comment reconfigurer les VNFs en les migrant vers un état optimal qui pourrait être calculé en avance ou inconnu. Dans cette thèse, nous avons étudié les deux cas en minimisant la durée d'interruption de service et la durée de migration des VNFs. Nous avons proposé des méthodes exactes et approchées. Parmi les méthodes exactes, nous citons deux modèles PLNE. Nous avons également proposé deux approches heuristiques, l'une basée sur la génération de colonnes et la deuxième utilisant la notion de "feedback arc set". L'objectif global de ce travail est donc de définir et d'étudier le problème de reconfiguration des VNFs dans le contexte du 5G network slicing, et de proposer des modèles mathématiques et des algorithmes efficaces pour résoudre les problèmes d'optimisation sous-jacents.

Mots-clés : 5G Network slicing, Reconfiguration des VNFs , Migration des VNFs , optimisation.

Abstract

In recent years, because of the unprecedented growth in the number of connected devices and mobile data, and the ongoing developments in technologies to address this enormous data demand, the fifth generation (5G) network has emerged. The forthcoming 5G architecture will be essentially based on Network Slicing (NS), which enables provide a flexible approach to realize the 5G vision. Thanks to the emerging Network Function Virtualization (NFV) concept, the network functions are decoupled from dedicated hardware devices and realized in the form of software. This offers more flexibility and agility in business operations.

Despite the advantages it brings, NFV raises some technical challenges, the reconfiguration problem is one of them. This problem, which is NP-Hard, consists in reallocating the Virtual Network Functions (VNFs) to fit the network changes, by transforming the current state of deployed services, e.g., the current placement of Virtual Machines (VM) that host VNFs, to another state that updates providers' objectives.

This PhD thesis investigates how to reconfigure the VNFs by migrating them to an optimal state that could be computed in advance or free placement. In this thesis, we studied both cases while minimizing the service interruption duration and the VNF migration duration. We have proposed exact and approximate methods. Among the exact methods, we cite two ILP models. We also proposed two heuristic approaches, one based on column generation and the second using the concept of "arc set feedback". The overall objective of this work is therefore to define and study the problem of VNF reconfiguration problem in the context of 5G network slicing, and propose mathematical models and efficient algorithms to solve the underlying optimization problems.

Keywords: 5G Network slicing, VNF reconfiguration, VNF migration, optimization.

Contents

Acknowledgement **5**

Résumé **7**

Abstract **9**

Liste des tableaux **15**

Liste des figures **18**

1 Introduction **19**

1.1 Thesis plan and contributions 21

1.2 List of publications 22

2 Industrial context **25**

2.1 5G Networks 26

2.2 Network Slicing 27

2.3 Background of NFV 29

2.3.1 What is NFV ? 29

2.3.2 NFV framework 30

2.3.3 NFV principles 31

2.4 VNF reconfiguration 32

2.4.1 What is the VNF reconfiguration problem? . . . 32
2.4.2 The VNF reconfiguration techniques . . . 36
2.4.3 A classification of VNF reconfiguration methods in literature . . . 42
2.5 Conclusion . . . 43

3 Overview of tools and methods from graph theory and optimization 45

3.1 Graph theory . . . 46
3.1.1 Definitions . . . 46
3.1.2 Topological Sorting . . . 47
3.1.3 Feedback Arc Set problem . . . 48
3.1.3.1 Heuristics for the feedback arc set problem . . . 48
3.2 Linear Programming . . . 50
3.2.1 Example of Linear Programming problems . . . 51
3.2.1.1 The knapsack problem . . . 51
3.2.2 Duality . . . 52
3.3 Column generation . . . 53
3.3.1 An example: path generation . . . 54

4 An ILP formulation 57

4.1 SYSTEM MODEL . . . 58
4.1.1 Problem definition . . . 58
4.1.2 Problem modeling . . . 60
4.2 Problem statement and formulation . . . 61
4.2.1 VNF reconfiguration problem: Problem statement . . . 61
4.2.2 Problem formulation . . . 61
4.2.2.1 Decision variables . . . 61
4.2.2.2 Problem constraints . . . 62

4.2.2.3 Objective function 64
4.3 Experimental results 65
4.3.1 Simulation Setup 65
4.3.1.1 Topology Dataset 65
4.3.1.2 VNF and slice Datasets 66
4.3.2 Evaluation Metrics 66
4.3.3 Simulation Result and Analysis 67
4.3.3.1 Evaluation according to the nature of slices 67
4.3.3.2 Evaluation according to the nature of datasets 68
4.4 Conclusion 70

5 A heuristic approach **73**
5.1 The problem modeling 74
5.2 Related work 76
5.3 Overview of the proposed approach 77
5.3.1 Heuristic used for the Feedback Arc Set problem 77
5.3.2 Parallel Topological Sorting algorithm 78
5.4 Experimental results 80
5.4.1 Evaluation Metrics 80
5.4.2 Simulation Result and Analysis 81
5.4.2.1 Acyclic graph 82
5.4.2.2 Cyclic graph 82
5.5 Conclusion 84

6 Column generation model **87**
6.1 Multiple Multidimensional Knapsack Problem 88
6.2 Solution methods for fixed final target 91

6.2.1 First compact formulation 91
6.2.2 Non-compact formulation 92
6.2.2.1 Master Problem 94
6.2.2.2 Pricing Problem 95
6.2.2.3 Achieving an integer solution 96
6.2.3 Second compact formulation 97
6.3 Solution methods for free final target 97
6.3.1 Column generation 98
6.3.2 Second ILP formulation 98
6.4 Numerical Results 99
6.4.1 Fixed final target 99
6.4.1.1 Cyclic graph 100
6.4.1.2 Acyclic graph 102
6.4.2 Free final target 103
6.4.2.1 Cyclic Graphs 104
6.4.2.2 Acyclic Graphs 105
6.5 Conclusion 107

7 Conclusion **109**

7.1 Thesis contributions: a summary 110
7.2 Future work and perspectives 111

List of Tables

2.1 Classification of VNF-R techniques 36

2.2 Objective-based classification of VNF-R approaches 42

2.3 Classification of VNF-R solutions 43

4.1 Table of Notations 62

4.2 Datasets of acyclic graphs 66

4.3 Datasets of cyclic graphs 66

4.4 Comparison between the ILP model and TS algorithm for acyclic graph 68

5.1 Datasets of acyclic graphs 82

5.2 Datasets of cyclic graphs 82

6.1 Notation 92

6.2 Datasets of cyclic and acyclic graphs 99

6.3 Interruption cost of cyclic graphs for all models 100

6.4 Interruption cost of acyclic graphs for all models 102

6.5 Interruption cost of cyclic graphs for all models 105

6.6 Interruption cost of acyclic graphs for all models 106

List of Figures

1.1 5G usage scenarios 20

2.1 Main drivers behind past cellular communications generations and 5G [1] 27

2.2 5G Network Slicing concept [2] 28

2.3 The vision of physical network functions (PNFs) to virtualized network functions (VNFs) 29

2.4 High-Level NFV Framework [3] 31

2.5 Illustration of VNF reconfiguration problem 33

3.1 Representations of graphs 46

3.2 Topological sorting example 47

3.3 How column generation works 54

4.1 Presentation of VNFs reconfiguration problem. **NSSF**: *Network Slice Selection Function*, **UDM**: *Unified Data Management*, **NRF**: *Network Repository Function*, **AUSF**: *Authentication Server Function*, **AMF**: *Access and Mobility Management Function*, **SMF**: *Session Management Function*, **PCF**: *Policy Control Function* 59

4.2 The graph representation of VNFs migrations 60

4.3 The reconfiguration plan of all VNFs migrations taking into consideration the SLA availability 65

4.4 The evaluation results of migration and interruption cost 67

4.5 The evaluation results of migration duration for the acyclic graph 69

4.6 Scalability evaluation results for cyclic graph 69
4.7 The evaluation results of migration duration for cyclic graph 70
4.8 The evaluation results of interruption duration for cyclic graph 70

5.1 Graph model 75
5.2 The proposed approach for cyclic graph 77
5.3 Overview of the FAS algorithm 79
5.4 Diagram of the proposed approach 80
5.5 Computation time for cyclic graph 83
5.6 The total interruption duration for the cyclic graph 83
5.7 The total migration duration for the cyclic graph 83
5.8 Example of the entire process 85

6.1 Graph representation 93
6.2 Interruption cost of cyclic graphs (Column Generation) 100
6.3 Execution time of cyclic graphs in seconds (Column Generation) 101
6.4 Execution time of cyclic graphs in seconds 101
6.5 Interruption cost of acyclic graphs (Column Generation) 102
6.6 Execution time of acyclic graphs in seconds (Column Generation) 102
6.7 Execution time of acyclic graphs in seconds 103
6.8 Interruption cost of cyclic graphs (Column Generation) 104
6.9 Execution time of cyclic graphs in seconds (Column Generation) 104
6.10 Execution time of cyclic graphs in seconds 105
6.11 Interruption cost of acyclic graphs (Column Generation) 105
6.12 Execution time of acyclic graphs in seconds (Column Generation) 106
6.13 Execution time of acyclic graphs in seconds 106

Chapter 1

Introduction

Nowadays, the world is experiencing an increased use and need of applications and new technologies, numerous devices and networks will be interconnected and traffic demand will constantly rise. New services emerge with various requirements, and many devices required faster speed and data transmission. 5G is at the crossroads of these new usages. It aims to better respond to the wide variety of needs and new demands through a unified technology that takes into account this diversity.

One of the visions of 5G is to have a multi-service network supporting a wide range of verticals with a varied set of performance and service requirements. The key to realize this vision is network slicing [4] which is defined by [5] as a concept for running multiple logical customized networks on a shared common infrastructure complying with agreed service level agreement (SLA) for different vertical industry customers and requested functionalities. In the 3rd Generation Partnership Project (3GPP) [6] [7], three standardized slice types are currently defined (see Fig. 1.1):

- mMTC – Massive Machine Type Communications: communications between a large number of objects with various quality of service needs. The objective of this category is to respond to the exponential increase of connected objects. It mainly encompasses all uses related to the Internet of Things. These services require extensive coverage, low energy consumption, and relatively restricted speeds.

- eMBB – Enhanced Mobile Broadband: concerns all applications and services that require an ever faster connection, for example, to allow viewing of ultra-high definition (8K) videos or wireless streaming of virtual or augmented reality applications.

- uRLLC – Ultra-reliable and Low Latency Communications: encompasses all the applications requiring extremely high reactivity as well as a very strong guarantee of message transmission. These needs are mainly found in transport (in case of risk of accident, for example), in medicine (telesurgery), etc.

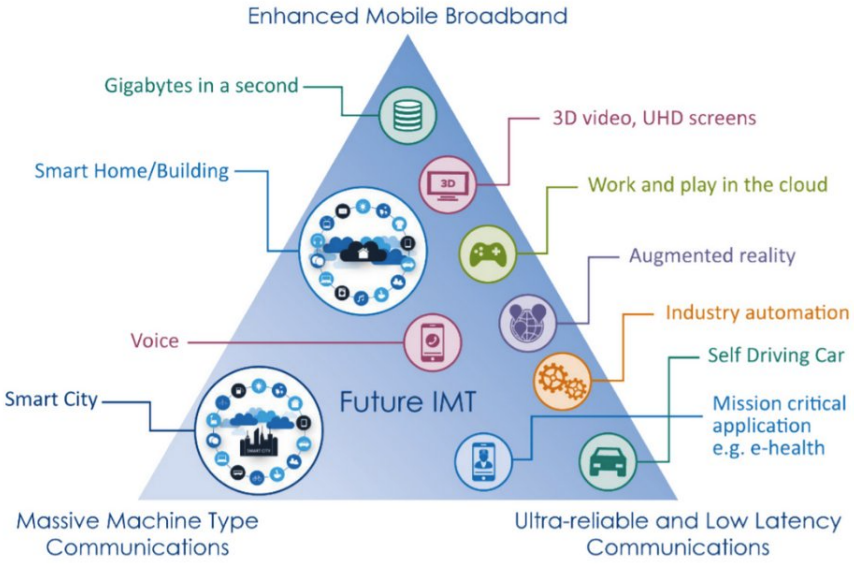


Figure 1.1: 5G usage scenarios

The network slicing is essentially based on Network Function Virtualization (NFV) which decouples hardware and network functions using virtualization technologies. A network slice is a connected set, or chain, of network functions, logically creating a dedicated virtual network that satisfies the specific requirements of a service [8]. By virtualizing entire categories of network node functions into modular units, NFV achieves greater scalability in communication and computing services. Dynamic slicing operations (such as the deployment of new slices, the ending of slices, etc) can impact the performance of other slices and make the placement of existing slice resources sub-optimal or inefficient. The operator should be able to reconfigure slices with minimal impact on ongoing services while respecting resource availability.

To provide high-quality and cost-efficient services, effective management of the 5G network is crucial. Through 5G management, most management tasks require reconfiguration of virtualized network functions. The VNF Reconfiguration has vital importance for both providers and users. In meeting the SLA, the poor estimation of the placement resources would lead to broken SLAs

and penalties. The client will be discouraged and the provider must compensate the client for SLA violations leading to significant economic losses. The VNF reconfiguration problem for the 5G core network is a challenging problem for the provider's resource-economic requirements.

The problem of VNF reconfiguration consists of reallocating the VNFs to adapt to the network changes, by transforming the current state of deployed services, e.g., the current placement of VMs or containers that host VNFs, to another state that updates providers' objective(s). There are three types of management strategies to address a reconfiguration problem: (i) VNFs migration (ii) Traffic re-direction and (iii) Scaling. In this work, we focus on the VM-based VNF reconfiguration problem regarding migration strategy.

1.1 Thesis plan and contributions

In this section, we present the plan of the thesis, summarise the main contributions, and list the published works.

In Chapter 2, we introduce the industrial context of our thesis, we overview the evolution of the mobile system, from 1G to 5G. Then, we present the concept of network slicing and its principles and introduce the background of NFV. Next, we highlight the VNF reconfiguration problem and the optimization techniques and algorithms used to solve it with a classification of the problem according to what exists in the literature.

In Chapter 3, we introduce different optimization tools and techniques that are used in this thesis. Firstly, we present an introduction to graph theory, secondly we introduce linear programming, and thirdly we present the concept of column generation based on linear programming.

In Chapter 4, we describe the first contribution of this PhD thesis. We define the problem of VNF reconfiguration in the context of 5G networks and propose an Integer Linear Programming (ILP) formulation to model it. The ILP model takes into consideration the type of VNF migration, the service interruption duration, and the VNF migration duration. The ILP finds a reconfiguration plan, consisting of a series of migrations that will relocate the VNFs from their current servers to those computed beforehand while minimizing the migration and interruption duration. We evaluate the proposed model according to the service importance taking into consideration the SLA availability metric, and the nature of the datasets. The content of Chapter 4 was presented at the International

Conference on Networks and published in its proceedings [9].

In Chapter 5, we propose an efficient heuristic for the Virtual Network Function (VNF) reconfiguration problem in the context of 5G network slicing. We rely on the topological sorting algorithm and the Feedback Arc Set (FAS) problem to build our heuristic which is an alternative to the exact solution already proposed in Chapter 4. The evaluation results show the good performance of the heuristic that solves the problem in a few seconds under a large network infrastructure.

In Chapter 6, we model the problem as Multiple Multidimensional Knapsacks Reconfiguration problems and formulate it through a non-compact model and the corresponding Column Generation algorithm. A new alternative compact ILP formulation is also presented. In addition, we propose an extension of the ILP formulation that enables reconfiguring the VNFs in the case where the destination is unknown. The VNFs migrate from the current state to another optimal state that the model ILP computes. The content presented in both Chapters 5 and 6 was respectively accepted at the International Workshop on Resilient Networks Design and Modeling [10] [11].

In Chapter 7, we conclude our work and present the directions to take to extend and improve our contributions to future research.

1.2 List of publications

Conference papers:

- Hanane Biallach, Mustapha Bouhtou, Dritan Nace. “L’optimisation de la reconfiguration des VNFs pour la gestion dynamique des slices dans les réseaux 5G”. 22e congrès annuel de la Société Française de Recherche Opérationnelle et d’Aide à la Décision (ROADEF), Apr 2021, Mulhouse, France.

- Hanane Biallach, Mustapha Bouhtou, and Dritan Nace, “Optimization of the virtual network function reconfiguration plan in 5G network slicing,” 21 International Conference on Networks (ICN), Apr 2022, Barcelona, Spain.

- Hanane Biallach, Mustapha Bouhtou, Dritan Nace. “An Efficient Heuristic for the Virtual Network Function Reconfiguration Problem”. 12th International Workshop on Resilient Networks Design and Modeling (RNDM), Sept 2022, Compiègne, France.

- Kristina Kumbria, Denis Demko, Dritan Nace, Artur Tomaszewski, Mustapha Bouhtou, Hanane Biallach. "VNFs reconfiguration in 5G networks". 12th International Workshop on Resilient Networks Design and Modeling (RNDM), Sept 2022, Compiègne, France.

Posters:

- Hanane Biallach, Mustapha Bouhtou, Dritan Nace, Sofiane Imadali. "Optimizing VNF reconfiguration for dynamic management of 5G network slicing". 10th International Conference on Operations Research and Enterprise Systems (ICORES), Feb 2021, Vienne (online), Austria.

Abstracts:

- Hanane Biallach, Mustapha Bouhtou, Dritan Nace, Sofiane Imadali. "Optimization and reconfiguration of 5G network slicing". 31st European Conference on Operational Research (EURO), July 2021, Athens, Greece.

Chapter 2

Industrial context

Contenu

2.1 5G Networks 26
2.2 Network Slicing 27
2.3 Background of NFV 29
2.3.1 What is NFV ? 29
2.3.2 NFV framework 30
2.3.3 NFV principles 31
2.4 VNF reconfiguration 32
2.4.1 What is the VNF reconfiguration problem? 32
2.4.2 The VNF reconfiguration techniques 36
2.4.3 A classification of VNF reconfiguration methods in literature 42
2.5 Conclusion 43

In this chapter, we introduce the industrial context of this PhD thesis. We present useful concepts needed to understand the scope of this work. Then, we describe the VNF reconfiguration problem together with different optimization methods used in the literature.

2.1 5G Networks

Mobile communication has transfigured the way people use to communicate each other to exchange information. Wireless communication including mobile generation technologies has a very tremendous growth in the past 50 years as given in fig 2.1. Since the 70s and over four decades, four generations of mobile wireless networks have emerged. In 1980 the mobile cellular era started, and since then mobile communications have undergone significant changes and experienced enormous growth. The first generation(1G) technology is a very basic voice analog phone system using circuited switched technology [12], that was introduced in the 1980s and continued until being replaced by 2G digital telecommunications. Compared to first-generation systems, second-generation (2G) systems offered low bitrates and allowed the support of multiple users by using digital multiple access technology, such as TDMA (time division multiple access) and CDMA (code division multiple access). Later in 2000, in order to offer voice, text, and data services, the third generation (3G) was introduced. In this technology, a true revolution took place where the network speed was increased up to 2Mpbs which made high-speed browsing, gaming, email, and other web services possible to a vast segment of the population and the transfer speed was four times greater than 2G. After the 3G transition, the next evolution took place with the introduction of mobile broadband and mobile data in the year 2010 when 4G was introduced. 4G is enabled by the LTE technology, which stands for Long Term Evolution. 4G networks use packet-switching to offer IP-based voice calls and text messages in addition to high-speed mobile data. Since the implementation of 4G networks and adoption by industrial firms, the number of mobile and Internet of Things (IoT) devices has drastically increased. In addition, the increase of users and their personal needs for bandwidth is causing congestion problems. The 4G network was not designed to be so adaptable and 5G is designed to meet these needs. 5G promises greater speed, capacity, and density of connected devices with less interference. It also delivers greater energy efficiency, greater mobility, and lower latency.

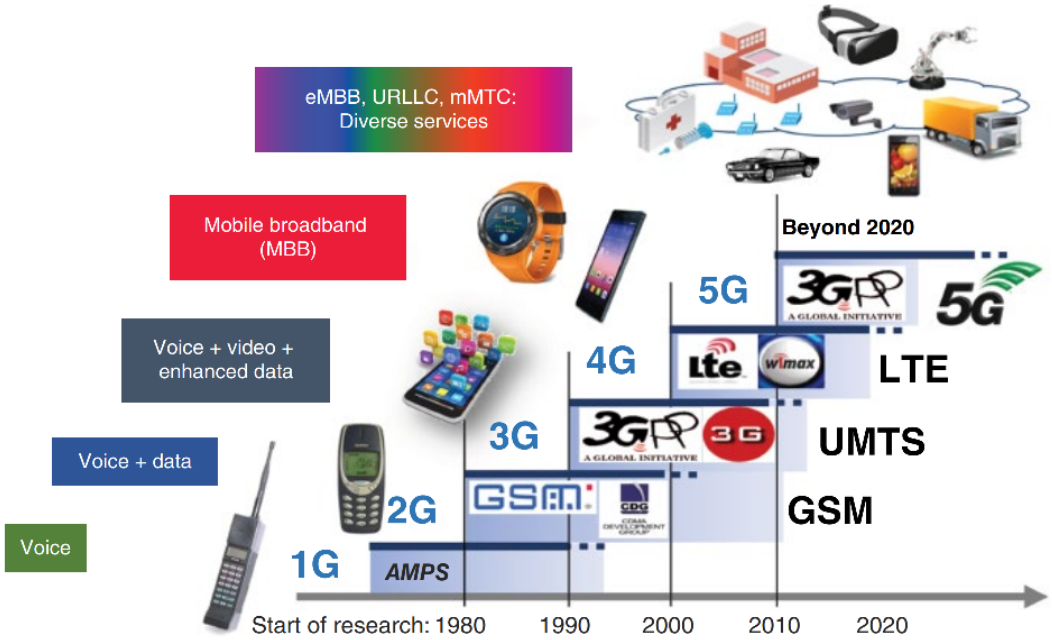


Figure 2.1: Main drivers behind past cellular communications generations and 5G [1]

2.2 Network Slicing

Unlike consumer services, vertical industries require networks that deliver deterministic services, with assured latency, jitter, and packet loss. Such requirements can hardly be met by 4G networks. Traditional networks should switch to 5G with its differentiated capabilities to fulfill varied requirements. Network slicing is the key to this change. With network slicing, a physical network can be sliced into logical networks that run on shared infrastructures while targeting different industries and applications.

An example of the network slicing architecture can be seen on Fig. 2.2. The network infrastructure is on the last layer and covers the access network, the transport network, and the core network. In this example, there are three slices, each of a different class and with different needs. Each slice has several entry points on the access network thanks to the Worldwide Interoperability for Microwave Access (WIMAX) infrastructure. Then, every slice passes through network functions on edge clouds in the transport network. Finally they connect to datacenters in the core network. Some parts of the network infrastructure are used by only one slice and others are shared by several of them.

In the context of 5G networks, Network slicing was introduced by NGMN (Next Generation Mobile

Network) in [13] as a paradigm that suggests partitioning the physical infrastructure into several virtual networks called slices, as presented in Fig. 2.2. Later, 3GPP [6] [7] defined network slicing as a technology that enables the creation of customized networks to provide optimal solutions for diverse market scenarios with diverse demand requirements. Network slicing builds on top of the following principles that shape the concept and related operations:

- Automation: is essential for network slicing because ultimately a plethora of dynamic network slices will need to be provisioned in real-time across physical, virtual, and cloud-based domains. The resulting complexity simply will not be manageable using manual processes.

- Isolation: is a major requirement that must be satisfied to operate parallel slices on a common shared physical network. Each slice has a specific service requirement that should be satisfied. Moreover, each slice must have an independent security function. Indeed, if two slices share the same resource and one slice had attacks or faults, the other slices must not have an impact.

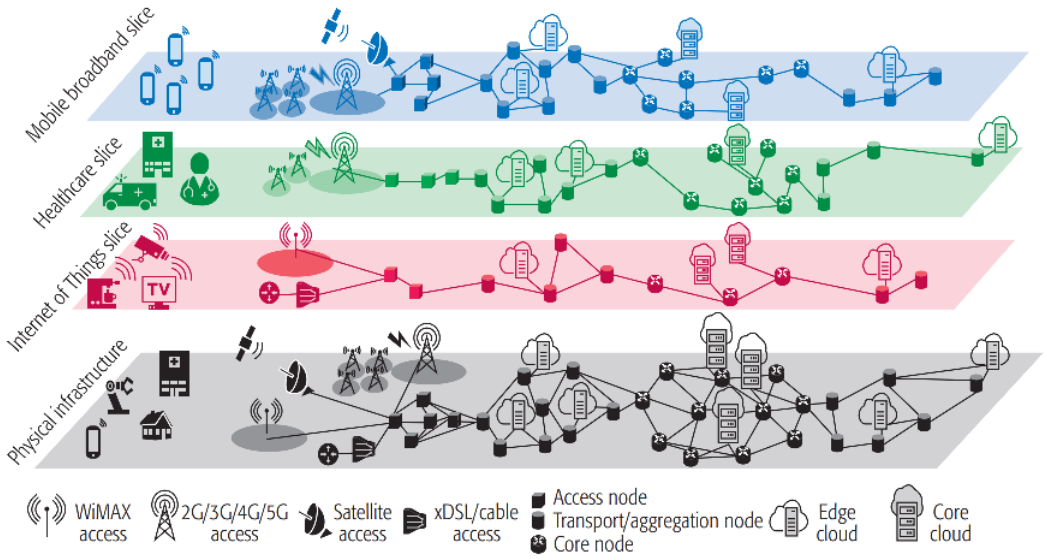


Figure 2.2: 5G Network Slicing concept [2]

- Elasticity: During the use of a slice, some needs may change and a slice must be elastic and must provide the possibility to change according to these needs. Elasticity can be achieved by shifting virtual network functions, scaling up and down allocated resources, and reprogramming the control and data element functionalities.

- End-to-end network slicing: is a major characteristic of the 5G System. It consists of RAN (Radio Access Network), Transport Network (TN), and Core Network (CN). This use case intends to demonstrate the modeling, orchestration, and assurance of a network slice which are implemented in the following with relevant standards.

2.3 Background of NFV

2.3.1 What is NFV ?

A Network Function (NF) is an element within a network with well defined external interfaces and functional behavior e.g. DHCP, Firewall, routing. Network Function Virtualization [14] [15] abstracts network functions, allowing them to be installed, controlled and manipulated by software. In traditional networks, all devices are deployed on the legacy. All network elements are enclosed boxes, and hardware cannot be shared. Each equipment requires additional hardware for increased capacity. With NFV, however, network elements are independent applications that are flexibly deployed on a unified infrastructure comprising standard servers, storage devices, and switches. In this way, software and hardware are decoupled, and the capacity for each application is increased or decreased by adding or reducing virtual resources.

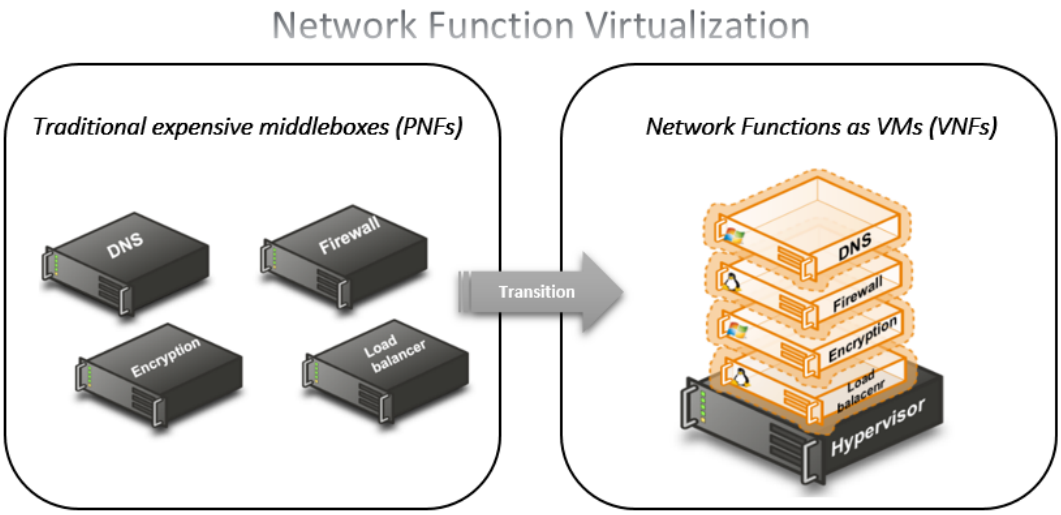


Figure 2.3: The vision of physical network functions (PNFs) to virtualized network functions (VNFs)

If NFV is implemented efficiently and effectively, it can provide a number of benefits compared to traditional networking approaches. The following are the most important potential benefits [16]:

- Reduced Capital Expenditures (CAPEX), by using commodity servers and switches, consolidating equipment, exploiting economies of scale, and supporting pay-as-you-grow models to eliminate wasteful overprovisioning.

- Reduced Operational Expenditures (OPEX), in terms of power consumption and space usage, by using commodity servers and switches, consolidating equipment, and exploiting economies of scale, and reduced network management and control expenses.

- The ability to innovate and roll out services quickly, reducing the time to deploy new networking services addressing changing requirements.

- Ease of interoperability thanks to standardized and open interfaces.

- Use of a single platform for different applications, users, and tenants. This allows network operators to share resources across services and across different customer bases.

- Provided agility and flexibility, by quickly scaling services to address changing demands.

2.3.2 NFV framework

The Fig. 2.4 shows a high-level view of the NFV framework. It consists of three domains of operation:

- **NFV Infrastructure (NFVI)**: The NFVI is the set of hardware and software components that build up the NFV environment on which NFV services are deployed, managed and executed. NFVI includes:

    1. *Hardware resources*: this includes computing hardware (such as RAM, CPU), storage hardware (such as disk storage, Network Attached Storage (NAS)), and network hardware (such as switches and routers).
    2. *Virtualization layer*: abstracts the hardware resources and allows to support of multiple VNFs by virtually sharing its resources. There are multiple open source and proprietary options for the virtualization layer (such as VMWare, Xen, and KVM).

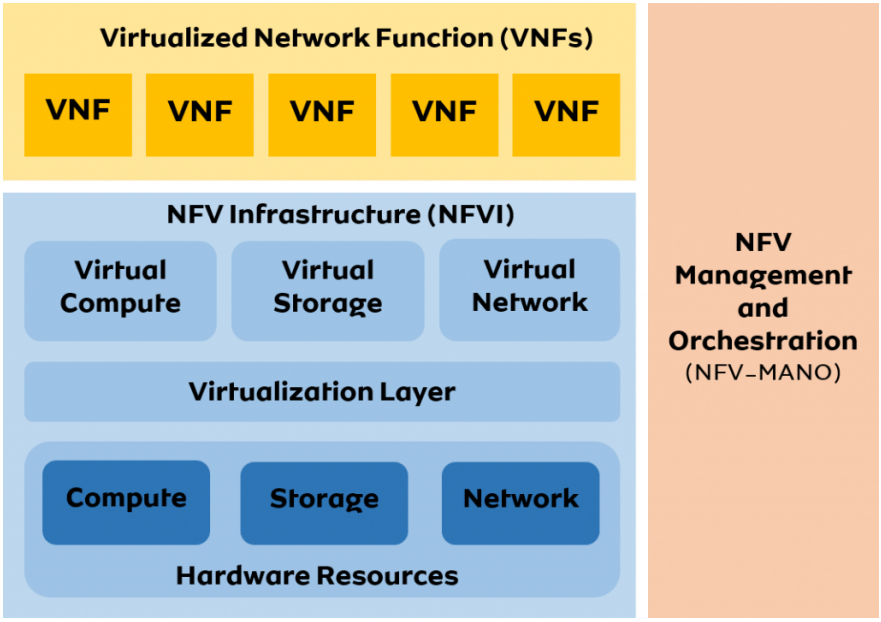


Figure 2.4: High-Level NFV Framework [3]

3. *Virtual infrastructure*: this includes virtual compute (Virtual Machines VMs), virtual storage, and virtual links.

- **Virtualized Network Functions (VNFs)**: represent virtualized instances of different network function, implemented in software, that run over the NFVI.

- **NFV Management and Orchestration (MANO)**: This involves deploying and managing the lifecycle of physical/software resources that support the infrastructure virtualization, and the lifecycle management of VNFs. NFV management and orchestration focus on all virtualization-specific management tasks necessary in the NFV framework [3].

2.3.3 NFV principles

Three key NFV principles are involved in creating practical Network Services (NS):

- Service chaining: Each NS requires a chain of network functions to be executed. This is referred to as Service Function Chaining (SFC in the IETF terminology), also known as Virtualized Network Functions Forwarding Graph (VNF-FG in the ETSI terminology). This means that there is a sequencing and specific connectivity between VNFs.

- VNF embedding: The process of deploying VNFs in the physical servers for the SFCs is called as VNF embedding. VNF embedding consists of node and link embedding. The orchestrator is responsible of the embedding stage; it is in charge of the management and orchestration of software resources and the virtualized hardware infrastructure to realize networking services.

- Distributed architecture: A VNF may be made up of one or more VNF components (VNFC), each of which implements a subset of the VNF's functionality. Each VNFC may be deployed in one or multiple instances. These instances may be deployed on separate, distributed hosts to provide scalability and redundancy.

2.4 VNF reconfiguration

2.4.1 What is the VNF reconfiguration problem?

In Network Service (NS), the traffic demand usually varies across time. This might affect the service performance and lead to breaking the SLA and penalties. Therefore, the orchestrator needs to manage and reconfigure the VNF resources to address the changes that occurred in the network. To define it simply, a reconfiguration is a reallocation of the NFV to adapt the utilization of network resources to the occurred changes and can have objectives such as minimizing the service interruptions or improving the QoS. Multiple types of reconfiguration can be done in the network:

- VNFs migration: There are two types of reconfiguration with VNFs migration. The first type is VNFs migration within a datacenter. As Cho et al. mentioned in [17], if two VNFs used one after the other are on the same node, then there is no communication delay between them. The second type is VNFs migration between datacenters. In this case, it may be associated with the re-routing of traffic between the new VNFs' positions. In [18], the authors consider the problem of reconfiguration as the problem of both rerouting traffic flows and improving the mapping of network functions onto nodes in the presence of dynamic traffic, with the objective of bringing the network back to a close to the optimal operating state, in terms of resource usage. In [19], the VNF reconfiguration problem is addressed by determining the migration of VNF and rerouting service paths between IoT networks and clouds that optimize the reconfiguration cost and the resource consumption, while satisfying the resource and QoS constraints.

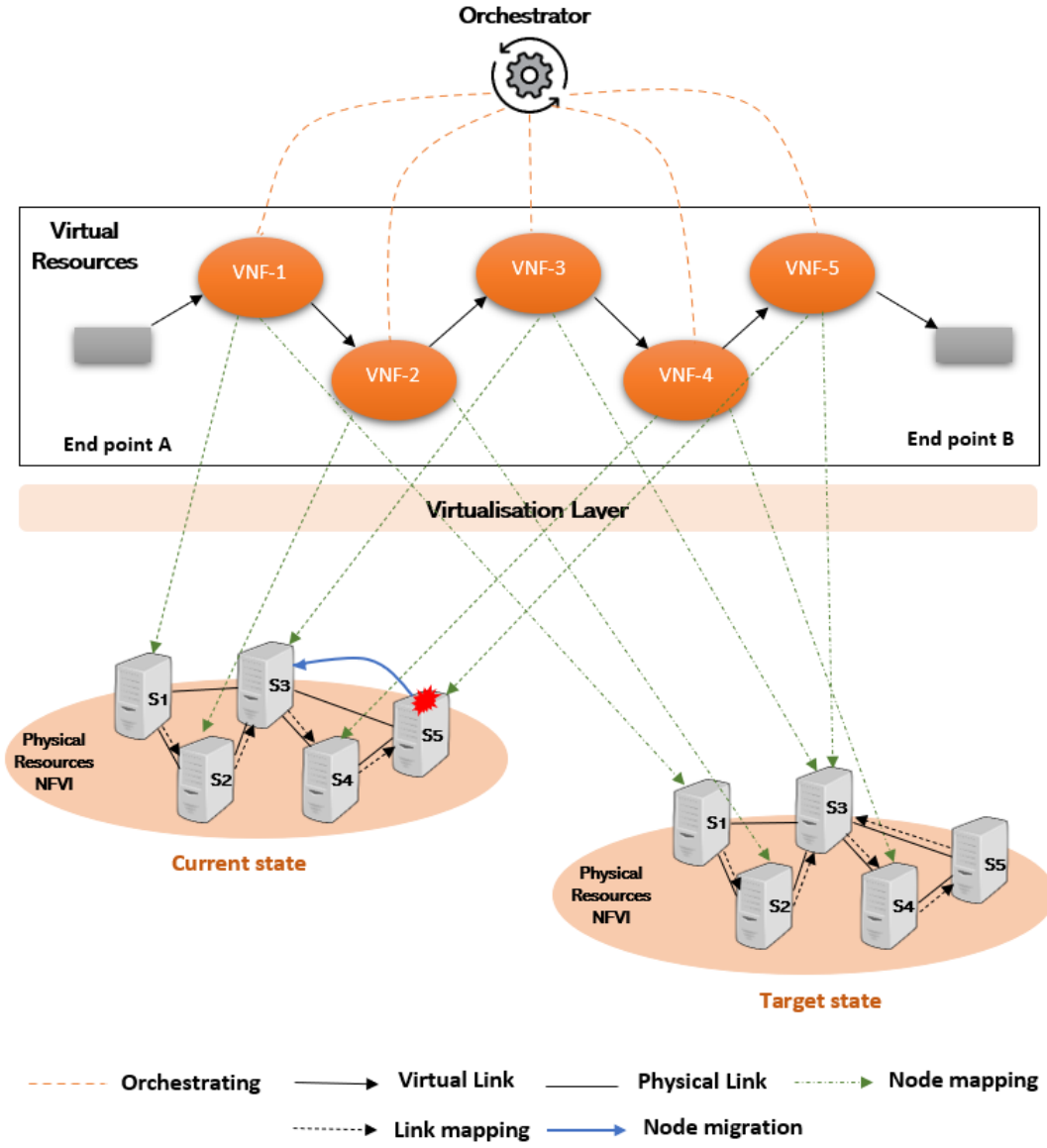


Figure 2.5: Illustration of VNF reconfiguration problem

- Traffic re-direction: This type of reconfiguration concerns flow re-routing. With this reconfiguration, a portion of the traffic is changed without impacting the network functions. If a request has to go through specific network functions, its traffic can be rerouted in-between but the functions will not migrate [18].

- Scaling: During high traffic periods, the virtual network, SFC or slice may need to be scaled. Stelios et al. [20] focus on the reconfiguration in terms of horizontal and vertical elasticity.

> Horizontal scaling (scale out/in) consists of adding/removing virtualized resources and vertical scaling (scale up/down) is reconfiguring the capacity/size of existing virtualized resources [21].

In our work, we focus on reconfiguration with VNFs migration. To help the reader grasp the problem of reconfiguration in terms of VNFs migration, Fig. 2.5 shows an example of the VNF reconfiguration problem of an end-to-end network service. The service is composed of a set of VNFs that together provide a specific functionality which is the SFC chain. The virtualization layer abstracts the physical resources and anchors the VNFs to the virtualized infrastructure. The SFC (virtual resources) is deployed in the VNF infrastructure (physical resources) presented by five servers in the "current state" which represents the initial state of our problem. The VNFs are implemented in virtual machines (VMs). At time t, we assume that a breakdown has occurred in server "S5". The VNF-5 deployed in S5 needs to be redeployed in order to ensure the continuity of the service. The strategy we take into consideration for the reconfiguration is migration. Here, the VNF5 is migrated to server "S3" in order to reach an optimal state presented by the "target state". We obtain the target state by using a simple model that calculates the VNF placement. The used model is not presented in this thesis.

In this example, we have just one VNF to migrate for reconfiguration. Imagine we have several VNFs migrations to perform, in this case, it will be difficult to know which one to migrate first while respecting the service requirements and available resources. Finding the order in which different VNFs can be migrated from source server to destination server is referred to as the VNF reconfiguration problem.

The VNF reconfiguration problem involves two tasks: (i) the first is identifying a new feasible placement of the VNF (and links) such that the service requirement can be satisfied. That is answering the question, where to place the set of VNFs that compose a network service in the most adequate way, with respect to the service provider goals, inside the physical network? (ii) the second task consists in identifying a sequence of VNF migrations (hence VM migrations) to pass from the current configuration, called *current state* (where VNFs are not optimally allocated) to the desired configuration, called *target state* (looking for optimality). This should be done while taking into consideration the different constraints (precedence and dependencies between the VNFs, hardware and network resources, etc). Here the question is, how to execute the migrations to reach a given placement? In this thesis, we

focus our attention on the first and second tasks, without taking into consideration the precedence constraints between VNFs and network.

- VNFs placement

In literature, the VNF placement problem is encountered in two forms:

- Initial VNFs placement: consists of mapping efficiently the initial SFC request onto the physical network [22] [23] [24].
- Dynamic VNFs placement: deals with the re-assignment of the physical network usage (the VNFs have to be re-allocated to adapt to dynamic changes occurred in the network) [25] [26]

The dynamic VNFs placement corresponds to the first task of the VNF reconfiguration problem.

- VNFs Migration

The movement of VNFs from one physical host to another physical host by transferring its consumption such as CPU, associated memory, and storage, is known as VNF Migration. The VNFs migration can be performed by different methods. Generally, there are two techniques of migration:

- Cold (non-live) migration: represents the process of moving VMs between servers by powering off the VM on the source server, moving it to the target server then powering it back up on the target server [27]. Migrating a suspended VM is named a cold migration.
- Hot (live) migration: consists of moving running VMs between the target and source servers without disconnecting the service or application. The live migration ensures minimal downtime for the VM. In addition, it facilitates fault management, low-level system maintenance, and reduction in energy consumption [28]. Among the different live migration approaches, there are two main algorithms, the pre-copy algorithm, and the post-copy algorithm. The terms pre- and post-copy refer to the transfer of memory pages. The pre-copy migration copies the memory state to the destination and then transfers the processor state. However, the post copy migration does the opposite, it sends the processor state to the destination and then transfers the VM's memory contents.

The VNFs migration is a reconfiguration strategy among others (scaling, traffic re-direction). In this thesis, we focus on the VNF reconfiguration that uses the migration strategy. In the example

presented in Fig. 2.5, the trigger for the migration is the breakdown. There are many other triggers or use cases for moving VNFs:

- Moving VNFs from servers that need upgrades, maintenance, or any other operation that could take longer.
- Moving VNFs when new demands arrive, for example, new SFCs need to be deployed.
- Moving VNFs from servers that need to reduce their energy consumption.
- Achieving high availability by moving VNFs to alternative servers when their current physical servers are failing or get inadvertently down.
- Achieving better performance by moving VNFs from one server to another, in order to avoid busy servers and ensure load balancing.

2.4.2 The VNF reconfiguration techniques

The VNF reconfiguration problem is a NP-Hard optimization problem. [29] demonstrates the NP-hardness of the reconfiguration problem in the context of distributed systems. In this section, the research on the VNF reconfiguration problem is summarized. Table 2.1 reports different works with regard to the problem. These works are classified based on the method they use, as discussed below.

Table 2.1: Classification of VNF-R techniques

Classification	**References**
Exact	[30] [31] [32] [33] [34] [35] [36] [37] [38] [39] [40]
Heuristics	[18] [32] [35] [41] [42] [43] [44] [45]
Meta-heuristics	[46] [47] [48]
Artificial intelligence	[19] [36] [49] [50] [51]

- Using exact methods

In literature, many research works to solve the problem of VNF reconfiguration by proposing an Integer Linear Programming (ILP) model [32] [33] [34] [35] [36] [37] [40]. Eramo et al. [32] formulated

the energy-aware reconfiguration of VNF as an ILP. Using cold migration, they proposed a migration policy that determines when and where to migrate VNF in response to changes to SFC request intensity. The objective is to minimize the total energy consumption given by the sum of the consolidation and migration energies. The same authors proposed in [33] a hot migration policy aiming at minimizing the total cost taking into account both the energy consumption and the reconfiguration costs characterizing the QoS degradation.

In [40], the authors studied the problem of SFC dynamic reconfiguration in edge clouds. They consider that the VNFs (and hence VNF Components (VNFCs)) have been already placed in the respective virtual resources on which they run. The resource requirements of each VNFC change over time with changes in traffic. They aimed to balance the trade-off optimization objective which is to maximize the service provider's revenue minus the total reconfiguration cost. The total reconfiguration cost is divided into the VNFs migration cost and bandwidth cost. The problem was formulated as an ILP model intended to exactly solve this problem.

In [34], Gausseran et al. treated the problem of reconfiguring the VNFs with the goal of bringing the network from a sub-optimal to an optimal operational state. They proposed an ILP model based on the make-before-break mechanism, in which a new path is set up before the old one is torn down. This mechanism allows reaching closer to optimal resource allocation while not interrupting the demands which have to be reconfigured. The authors showed that the model allows to lower the network cost and increase the acceptance rate.

In [35], an ILP model was formulated to solve the problem of deployment and reconfiguration of VNFs for dynamic situations. The authors tried to jointly optimize the deployment and the reconfiguration of new users' services while considering the trade-off between resource consumption and operational overhead. While in [36], the authors consider a trade-off between the minimization of the latency, Service Level Agreement (SLA) violation cost, hardware utilization, and VNF reconfiguration cost. In [37], the problem of VNF reconfiguration is addressed in the context of cloud computing. The authors treated the problem by answering the questions of when to reconfigure, which VNFs should be reconfigured, and where to host them. To deal with this issue they proposed a two-level runtime reconfiguration mechanism to automate the operations with the objective of reducing VNF migration and minimizing the total migration time.

A divide-and-conquer approach is proposed in [38], in order to reconfigure 5G network slices. The

proposed model finds a sequence of feasible migrations, that satisfies the SLA without overloading the network, from the current configuration to the new configuration while taking into consideration the bandwidth and latency constraints between the network functions of network slices.

The reconfiguration problem is treated in [30] [31] [39] using constraint programming, which is an approach to model and solve combinatorial problems. In [30] [31] authors proposed a VM scheduler named BtrPlace that uses constraint programming (CP) to model the position of the VMs and the migrations. From its inputs, Btrplace models a core Reconfiguration Problem (RP), i.e. a minimal reconfiguration algorithm that manipulates servers and VMs through actions. To use CP, a problem is modeled as a Constraint Satisfaction Problem (CSP), comprising a set of variables, a set of domains representing the possible values for each variable, and a set of constraints that represent the required relations between the values and the variables. A solution for a CSP is a variable assignment (a value for each variable) that simultaneously satisfies the constraints. To solve CSPs, BtrPlace uses the Choco library [52]. Constraint programming is also used in [39] for dynamic consolidation, which is an approach that migrates tasks within a cluster as their computational requirements change, both to reduce the number of nodes that need to be active and to eliminate temporary overload situations. The type of migrations taken into account is live (hot) migrations.

- Using heuristic methods

The problem of VM-based reconfiguration is tackled in [43] [44] [41] [45] using greedy heuristics. In [43], the problem of virtual network function reconfiguration was treated with the objective of minimizing the number of overloaded links and reducing the cost of reconfiguration. To deal with this problem, the authors proposed a greedy Virtual Network Reconfiguration algorithm (VNR) that minimizes the rejection rate of virtual network requests. In [44] greedy heuristics are provided for managing the resource provisioning of reliable virtual networks (VN) in the cloud. This includes handling the placement of VNFs while providing availability guarantees, as well as reconfiguring their placement as their request changes over time. It consists of two modules, JENA: a tabu-based availability-aware resource allocation (embedding) module for virtual networks that achieves availability guarantees, and ARES: a reliable reconfiguration module to adapt the embedding of hosted services as they scale.

A greedy heuristic algorithm is proposed in [41] to reconfigure the live migrations of a given set of VMs so that all VMs are migrated to their final target. The goal is to calculate a migration plan that minimizes the total migration time and the total migration downtime of the VMs. The authors

took into account the datacenter network topology using a weighted undirected graph. The proposed heuristic calculates a migration cost for each VNF based on: the CPU and memory usage of the VNF, the rate of modification of the memory pages as well as the amount of traffic exchanged with other VNFs. In [45], the authors presented VScheduler, a system that dynamically adjusts the resource configuration of virtual machines, including the amount of virtual resources and a new mapping of virtual machines on physical nodes. In order to find a good solution close to the optimal, a greedy heuristic was employed to dynamically redistribute virtual machines on a cluster of physical nodes. A greedy heuristic is also proposed in [53] in the context of embedded systems. The authors proposed a GRASP algorithm that addresses the joint placement and routing of networks of processes, which is a static reconfiguration of tasks from a DPN (represents the network of processes) onto the network of clusters. The proposed algorithm gives good solution quality with an acceptable computation time.

Column generation models are proposed in [35] and [18] in order to solve larger instances. In [35], the dynamic service reconfiguration problem was addressed. First, the authors solved the problem exactly using an ILP model, as mentioned earlier. Then, to reduce the time complexity they designed a column generation (CG) model for optimization. The results showed that the CG-based approach outperformed the benchmark algorithm in terms of the service provider's profit from service provisioning. In [18], a scalable algorithm relying on the column generation technique was modeled to solve the problem of network slice reconfiguration, while avoiding QoS degradation. The proposed algorithm reconfigures a given set of network slices from an initial routing and placement of network functions to another solution that improves the usage of the network resources, both in terms of links and VNFs.

A dynamic programming algorithm is proposed by Eramo et al. in [32] using a Viterbi algorithm to find the best VNF reconfiguration for traffic arriving at every time instance. As defined in [54], a Viterbi algorithm is a recursive optimal solution to the problem of estimating the state sequence of a discrete-time finite-state Markov process. The proposed heuristic is able to determine when and where to migrate VNFs in response to network changes, with low computational complexity.

- Using meta-heuristic methods

Saving energy consumption remains one of the interesting challenges of VNF. [48] and [47] treated the problem of reconfiguration regarding to energy consumption. Both papers provide genetic algorithm models to generate high-quality solutions to optimization and search problems. In [48], the authors proposed an energy-aware service and reconfiguration algorithm that allows users to meet

their service latency requirements while minimizing energy consumption at the same time. The proposed algorithm also reconfigures the SFC path when the energy consumption of idle servers exceeds a given threshold of the total energy consumption, which results in reducing total energy consumption. For the same objective of reducing the consumed energy, Haibo et al. [47] tried to conserve the power consumption in the large-scale datacenter. To solve this issue, they proposed a standard genetic algorithm-based online self-reconfiguration approach (GABA), that reconfigures the number of VMs for different applications and their physical locations according to time-varying resource requirements and environmental conditions. The genetic algorithm is adopted to efficiently find the optimal reconfiguration policy. The resource utilization of large-scale cloud computing data centers can thus be improved and their energy consumption can be greatly conserved.

A tabu-search algorithm is used both in [42] [40]. Authors of [42] tried to answer the VNF reconfiguration problem by mapping and scheduling VNFs of a given service onto virtual networks. To do so, they proposed three greedy algorithms and a tabu search-based heuristic. As they mentioned, the tabu search-based algorithm performs better than the greedy ones. However, the algorithms proposed do not consider the links between physical/virtual nodes, and consequently, the link delays for transferring a given function from one node to another are considered to be negligible. In [40], a tabu-search heuristic is proposed as well. The proposed algorithm can balance the trade-off between the service provider's revenue and reconfiguration cost.

A genetic algorithm is also proposed in [46], to find the optimal allocation of virtual machines in a multi-tier distributed environment. The authors considered a general model for virtual machines resource allocation where each virtual machine and each physical host is described by a multi-dimensional resource vector. Resources are both quantitative and qualitative. The genetic algorithm appears to perform well in finding an approximate solution and the efficiency obtained could make the genetic algorithm feasible in a dynamic allocation scenario.

- Using Artificial intelligence methods

The reconfiguration of network slices is investigated in [50] with aim of minimizing long-term resource consumption by exploiting Deep Reinforcement Learning (DRL). The authors addressed the resource reconfiguration under flow fluctuation within a network slice, the proposed model predicts the future requirement of the slice thus to avoid unnecessary reconfigurations. The DRL is also used in [51] in order to provide a better quality of service (QoS) for diverse applications. The authors

proposed a framework integrated into the Software Defined Network (SDN) control plane of the Data Center Networks (DCNs), implementing real-time and automatic DCN reconfiguration.

In addition to ILP, [36] also proposed a Machine Learning approach for Efficient reconfiguration of VNFs called MAPLE. The proposed approach works in a proactive fashion to reduce both the setup latency and complexity of the VNF reconfiguration processes. To do so, the k-medoids clustering technique, which is praised for its robustness to noise and outliers compared to other clustering techniques (e.g., means), is employed [55]. However, since k-medoids suffer from several limitations in terms of their sensitivity to local optima and their time-consuming running time, the authors incorporated a statistical technique that optimizes the selection of the initial set of medoids into the proposed clustering approach to enhance the clustering performance and clusters quality. Therefore, the authors modeled the VNF reconfiguration problem as an Integer Linear Programming (ILP) problem which considers a tradeoff among the minimization of the latency, SLA violation cost, hardware resource utilization, and VNF readjustment cost. Then, machine learning-based placement and reconfiguration algorithm were designed that intelligently eliminate some cost functions from the proposed ILP to boost its feasibility in a large-scale network.

The dynamic reconfiguration of computational resources assigned to virtual network functions plays a key role in the deployment of management of future service infrastructure. In [49] [19], the problem of service function reconfiguration is studied. Hirayama et al. [49] investigated the problem of VNF reconfiguration, using a migration strategy, by utilizing an encoder-decoder the Recurrent Neural Network (RNN). The RNN approach is effective, especially when sequential input data is available. Therefore, it is also suitable for the dynamic VNF migration technique by utilizing the time-series data of resource demands of network functions. In [19], the dynamic VNF reconfiguration is treated in the context of Internet-of-Things (IoT). In addition to ILP, the authors solved the problem through a deep learning approach while guaranteeing the QoS and resource constraints. Regarding the NP-hardness of the proposed ILP model, the authors proposed a deep Dyna-Q (DDQ)-based approach to efficiently reconfigure SFCs in the IoT network. the proposed approach uses two algorithms that are used back-to-back resulting in the graph neural network (GNN)-based VNFI resource prediction and the Dyna-Q-based SFC reconfiguration.

Table 2.2: Objective-based classification of VNF-R approaches

Energy	Energy Consumption	[32] [33] [56] [47] [48]
Cost	Reconfiguration Cost	[18] [19] [36] [37] [44] [45] [50] [57] [56]
	Migration Cost	[19] [30] [31] [43] [47]
Resource Usage	Hardware usage	[30] [31] [36] [39] [43] [48] [49] [57] [53]
	Network Usage	[19] [30] [31] [34] [35] [38] [41] [42] [45] [46] [50] [57] [53]
QoS	Latency	[36] [38] [47] [51]
	Overhead	[35] [44]
	Reconfiguration Delay	[18]
	Congestion	[46]

2.4.3 A classification of VNF reconfiguration methods in literature

In this section, we classified the nature of VNF reconfiguration solutions, proposed in the literature, into two categories (see Table 2.3):

- Online VNF-R (dynamic): In this case, the reconfiguration decisions are made in run-time. The datacenter makes the reconfiguration decisions based on its current state to satisfy the new arrival demands and take into account different constraints. The new arrival demands are not known, so the datacenter makes an immediate reconfiguration decision every time a demand arrives [35] [47].

- Offline VNF-R (static): This case requires prior knowledge about arrival demands. The reconfiguration decisions are made to satisfy the demands and constraints known beforehand. The datacenter is not able to update the new changes that occurred over time.

The VNF-R solutions are categorized according to their target objectives (see Table 2.2):

- Energy consumption: can be represented by minimization of power consumption.

- Cost optimization: can be expressed in terms of reconfiguration cost and migration cost.

- Resource usage: can be represented by hardware usage (CPU, RAM, etc) and network usage (bandwidth, etc).

- QoS Optimisation: can be expressed in terms of latency, overhead, reconfiguration delay, and congestion.

Table 2.3: Classification of VNF-R solutions

Type of reconfiguration	**References**
Online VNF-R	[18] [19] [32] [33] [34] [35] [36] [39] [42] [41] [44] [45] [47] [50] [49] [51] [57]
Offline VNF-R	[30] [31] [32] [37] [38] [41] [43] [46] [48] [53]

2.5 Conclusion

In this chapter, we defined the concept of network function virtualization (NFV) followed by a description of its architecture and principles. Then we presented a definition of the VNF reconfiguration problem and its relationship with the concepts of VNF placement and migration. Moreover, we highlighted the optimization techniques and algorithms used to solve the problem of VNF reconfiguration with a classification of the problem according to what exists in the literature.

Chapter 3

Overview of tools and methods from graph theory and optimization

Contenu

3.1 Graph theory 46
3.1.1 Definitions 46
3.1.2 Topological Sorting 47
3.1.3 Feedback Arc Set problem 48
3.2 Linear Programming 50
3.2.1 Example of Linear Programming problems 51
3.2.2 Duality 52
3.3 Column generation 53
3.3.1 An example: path generation 54

In this chapter, we describe some theoretical aspects that are fundamental to a better understanding of our work. We begin by describing some concepts of the graph theory and some classical problems that are used within our proposed approaches. Then we describe some exact approaches and the column generation approach used to address optimization problems.

3.1 Graph theory

Graphs are widely used in computer science and applied mathematics. They are vastly used to describe different types of networks and scheduling processes for example. This section formalizes some concepts of graph theory to better represent 5G networks.

3.1.1 Definitions

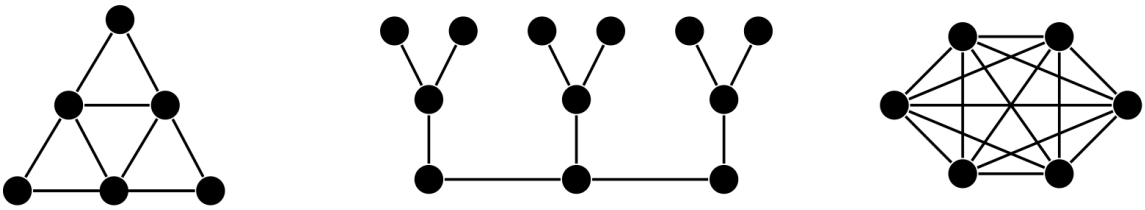


Figure 3.1: Representations of graphs

Graphs are networks of points and lines, like the ones shown in Figure 3.1. As described in [58] , a graph G consists of a finite nonempty set V of objects called *vertices* and a set E of 2-element subsets of V called *edges*. The sets V and E are the *vertex set* and *edge set* of G, respectively. So a graph G is a pair (actually an ordered pair) of two sets V and E that we can write it as $G = (V, E)$.

A directed graph, also called a digraph, is a graph in which the edges have a direction. The set E of a directed graph, for example, is therefore composed of pairs (u, v), where u and v are called *origin* and *destination* respectively. We call the *size* of the graph the cardinality of the set E, that is, the number of arcs of the graph. Similarly, we call *order* the cardinality of the set V, that is, the number of vertices of the graph.

A graph whose vertices are arranged in a row, is called a path graph (or often just called a path). Formally, the path P_n has vertex set $\{v1, v2, ...vn\}$ and edge set $\{v_i v_{i+1} : i = 1, 2, ...n\}$. If we arrange vertices around a circle or polygon, we have a cycle graph (often just called a cycle). Another way to

think of a cycle is as a path where the two ends of the path are connected up. Formally, the cycle C_n has vertex set $\{v1, v2, ...vn\}$ and edge set $\{v_i v_{i+1} : i = 1, 2, ...n\} \cup \{v_n v_1\}$. In this context, a graph with no cycles in it is known as an *acyclic graph*, and a graph with at least one cycle is known as *cyclic graph*. A complete graph is a simple graph in which every vertex is adjacent to every other vertex. Formally, a complete graph K_n has vertex set $\{v1, v2, ...vn\}$ and edge set $\{v_i v_j : 1 \leq i < j \leq n\}$.

3.1.2 Topological Sorting

Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of nodes such that for every directed arc $(u; v)$, node u comes before v in the ordering. Topological Sorting for a graph is not possible if the graph is not a DAG. For example, see Figure 3.2 , a topological sorting of the following graph is [7, 5, 3, 1, 4, 2, 0, 6]. There can be more than one topological sorting for a graph. For example, another topological sorting of the following graph is [3, 5, 7, 0, 1, 2, 6, 4].

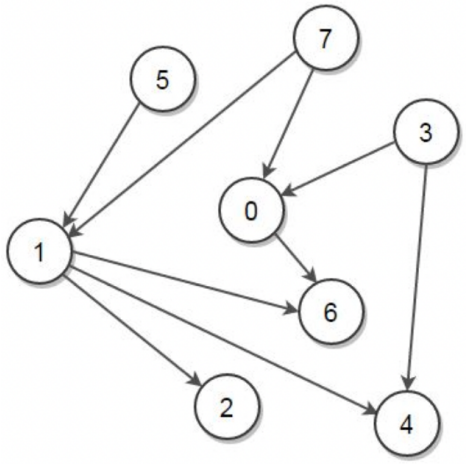


Figure 3.2: Topological sorting example

Kahn [59] and depth-first search [60] algorithms are two classical algorithms that find a topological sorting for a given DAG. Both algorithms have running time linear in the number of nodes plus the number of arcs O($|V| + |E|$) where $|V|$ denotes the number of nodes in the graph and $|E|$ the number of arcs in the graph.

3.1.3 Feedback Arc Set problem

A feedback arc set is a subset of edges containing at least one edge of every cycle in a directed graph. In other words, removing the edges in the feedback arc set from the graph makes the remaining graph a directed acyclic graph. A feedback arc set S is minimal if the reinsertion of any edge $s \in S$ to the directed acyclic graph induces a cycle. If the edges in a minimal feedback arc set are reversed rather than removed from the original graph, then the graph also becomes acyclic [61].

The feedback arc set problem is on the list of Richard M. Karp's 21 NP-complete problems [62]. The minimum feedback arc set problem is an NP-hard problem on graphs that seeks a minimum set of arcs that leaves a directed graph G acyclic when removed.

3.1.3.1 Heuristics for the feedback arc set problem

In this section, we present some heuristics for the feedback arc set problem. We consider $G = (V; E)$ as a directed graph with $|V| = n$ and $|E| = m$.

- **The minimum set cover problem approach**

The greedy heuristic consists of gradually building the feedback arc set by always picking that arc as the next element that, when removed, destroys most of the remaining simple cycles. This heuristic is known as the greedy heuristic for the minimum set cover problem, with an $O(1+log(d))$ approximation factor, where d is the maximum cardinality of any subset. The weakness of this heuristic is that even sparse graphs can have $\Omega(2n)$ simple cycles [63].

- **Greedy local heuristics**

As explained above, counting all the simple cycles makes the greedy approach less efficient. Other heuristics, based on local information only and making greedy choices without enumerating all simple cycles, exists. The feedback arc set is built up iteratively. At each step, the input graph is simplified before removing an arc or multiple arcs. This simplification can include splitting into Strongly Connected Components (SCCs) and then dropping the trivial SCCs (the ones consisting of a single node).

After simplification, the algorithm looks for a node in the remaining graph where many simple cycles are likely to be destroyed when one or a few arcs of that node are removed. For example, a node in a SCC with a single incoming arc but with many outgoing arcs is a good candidate: removing its single incoming arc breaks all the cycles that pass through that node, and the number of destroyed simple cycles is at least the out-degree of that node (each outgoing arc must participate in at least one simple cycle in a SCC by definition). This intuition is behind the greedy score functions: a node gets a higher score if it is more "asymmetric" regarding its in- and out-degrees. Such score functions are, for example:

$$score(i) = |d_i^{in} - d_i^{out}|$$

or

$$score(i) = max(\frac{d_i^{in}}{d_i^{out}}, \frac{d_i^{out}}{d_i^{in}})$$

Where $score(i)$ is the score of node i, d^in_i and d^out_i are the in- and out-degree of the node i, respectively. There are weighted variants of these functions that can be used if the input is a weighted graph. The node with the highest score is selected, then all of its in- or outgoing arcs are removed, whichever arc set has the smallest cardinality. The algorithm continues with the simplification. The heuristic terminates when there are no arcs left [64] [65].

- **Sorting heuristics**

Given an arbitrary ordering of the nodes of G, all the arcs can be categorized as either forward or backward arcs depending on whether the terminal node (source) of the arc appears after the initial node (destination) of the same arc or before. In the former case, the arc is a forward arc (it is pointing forward in the ordering); in the latter case, it is a backward arc. The set of backward arcs is selected as the feedback arc set. The sorting heuristics view the minimum feedback arc set problem as an ordering problem: they try to find the minimum cost ordering by sorting the nodes appropriately. Various sorting heuristics exist, for example in [66].

3.2 Linear Programming

Linear programming, also called linear optimization, is, as the second name suggests, a method for solving optimization problems. Since 1947 with the introduction of the simplex algorithm by Dantzig [67] (the first practical approach to solving linear programs), linear programming has become one of the most widely used methods for solving optimization problems. Optimization can be as the mathematical branch intended to handle a complex decision problem, involving the selection of values for a number of interrelated variables, by focusing attention on a single objective designed to quantify performance and measure the quality of the decision. The objective is maximizing (or minimizing, depending on the formulation) subject to the constraints that may limit the selection of decision variable values. All dependencies expressed through constraints or objective function are assumed linear.

In a linear program there can be distinguished:

1. Parameters, which are the data describing the instance of the problem which are considered as known given values.
2. Decision variables are the unknowns that the user needs to determine.
3. Objective (or optimization) function. This is used to express the objective (minimization or maximization) in terms of the decision variables.
4. Constraints are functions that express the requirements on the relationships between the decision variables.
5. Domain of variables allows limiting the set of decision variables as part of continuous/integer/positive... numbers.

Linear programming (LP) is widely used to solve optimization problems in operations research such as scheduling, flow routing, resource allocation, resource management, etc. In a LP, all variables must be fractional, however, there are many problems that require integer or binary variables. For example, to model a delivery vehicle scheduling problem, it is impossible to use a fraction of a vehicle, when a truck is sent on delivery it is sent entirely. When all variables are binaries or integers the problem is an ILP (Integer Linear Program). When there is a mix of fractionals and integers/binaries

variables the problem is a MILP (Mixed Integer Linear Program) but we will refer to it as ILP during this thesis to keep the nomenclature simple.

Linear programs are written in the form:

$$\text{Minimise/Maximise} \quad \sum_{i=1}^{n} c_i x_i \tag{1}$$

$$\text{Subject To:} \quad \sum_{i=1}^{n} a_{j,i} x_i \begin{cases} \leqslant \\ = \\ \geqslant \end{cases} b_j, \quad j \in [1, m] \tag{2}$$

$$x_i \begin{cases} \leqslant \\ \geqslant \end{cases} 0, \quad i \in [1, n] \tag{3}$$

Where n is the number of variables and m the number of constraints. $X = \{x1, x2, ..., xn\}$ is the set of decision variables. $C = \{c1, c2, ..., cn\}$ is the set corresponding to the coefficients of the variables in the objective function, it may be a cost to minimise or a profit to maximise. For every linear constraint $j \in [1, m]$ there is a set of coefficient $a_{j,i}$ for every variables x_i and b_j is the *right-hand-side* of equation j.

Different classes of solutions (i.e., classes of values for the decision variables) are identified for a Linear Programming model:

- **Feasible solution:** is a solution for which all the constraints are satisfied;
- **Infeasible solution:** is a solution for which at least one constraint is violated;
- **Optimal solution:** is a feasible solution that has the most favorable value of the objective function.

Linear Programming models can be solved graphically in the simplest cases. In more complex cases other algorithms such as Simplex or interior-point are used.

3.2.1 Example of Linear Programming problems

3.2.1.1 The knapsack problem

The knapsack problem (KP) [68] has a significant place in the study of integer programming models with binary variables. In the knapsack problem, one needs to pack a set of items (I), with given values

(v) and sizes (w) (such as weights or volumes), into a container with a maximum capacity (C). If the total size of the items exceeds the capacity, we can't pack them all. In that case, the problem is to choose the a subset of the items of the maximum total value that will fit in the container. The decision variables are represented by binary variables x_i taking value 1 if item i is selected and packed into the container and 0 otherwise. The objective function is $(\sum_{i \in I} v_i x_i)$ such that $(\sum_{i \in I} w_i x_i \leq C)$.

3.2.2 Duality

One of the key properties of LP used in this section is Duality. For each LP written in standard form, called the Primal, there is another LP called the Dual. The objective of the dual is the opposite of the primal one, i.e. if the primal is a minimization problem, then the dual is a maximization one, and vice-versa. For every variable in the primal, there is a constraint in the dual. And for every constraint in the dual, there is a variable in the dual. The dual problem of the dual is primal.

The dual problem (left) of the primal (right) is modeled as follows:

$$\begin{array}{lll} \text{Minimise} & b^T y & \\ \text{Subject To:} & A^T y \geqslant c & \\ & y \geqslant 0 & \end{array} \qquad \begin{array}{lll} \text{Maximise} & c^T x & \\ \text{Subject To:} & Ax \leqslant b & \\ & x \geqslant 0 & \end{array}$$

- $c = \begin{pmatrix} c_1 \\ \dots \\ c_n \end{pmatrix}, x = \begin{pmatrix} x_1 \\ \dots \\ x_n \end{pmatrix}$ are column vectors of size n.
- $b = \begin{pmatrix} b_1 \\ \dots \\ b_m \end{pmatrix}$ is a column vector of size m.
- $A = \begin{pmatrix} a_{1,1} & \dots & a_{1,n} \\ \dots & \dots & \dots \\ a_{m,1} & \dots & a_{m,n} \end{pmatrix}$ is a matrix of size $m \times n$.
- c^T represents the transpose of vector c.

where y is the vector variables of size m of the dual problem.

There are several rules for transforming the constraints and bounds of a primal maximization problem to its dual:

- For an inferiority constraint and a non-negativity bound, the dual has a superiority constraint and a non-negativity bound.
- For an equality constraint and a non-negativity bound, the dual has a superiority constraint and no bound.
- For an inferiority constraint with no bound, the dual has an equality constraint and a non-negativity bound.

The comprehension of the duality in LP is important to understand the functioning of the column Generation (CG).

3.3 Column generation

The method was initially proposed by Ford and Fulkerson in 1958. Formally, column generation is a way of solving a linear programming (LP) problem that adds columns (i.e. variables) during the pricing phase of solving the problem. In that context, many researchers have observed that column generation is a very powerful technique for solving a wide range of industrial problems to optimality or to near-optimality. Among the others, Gilmore and Gomory have demonstrated its effectiveness in a cutting stock problem. Column generation relies on the fact that the solver does not need to access to all the variables of the problem simultaneously. In fact, the main advantage of column generation is that not all variables need to be put in formulation (the same principle as the constraint generation algorithm). Instead, the problem is first formulated as a restricted master problem with few variables, and new variables are brought into the basis as needed. Meaning that if a column with a positive (resp. negative) reduced cost for maximization (resp. minimization) problem can be found, it is added to the Master and this process is repeated until no more columns can be added to the Master (see Figure 3.3).

The column generation algorithm works as follows:

1. Start with a subset of columns forming the master problem.

2. Solve the master problem and identify the dual coefficients corresponding to constraints of Master.

3. Write the column generator algorithm: pose the components of a potential column as variables and dual coefficients (found at step 1) as parameters and write down an LP problem having as objective function the reduced cost of the column and as constraints the conditions that variables should satisfy to formulate a valid column. Solve the problem.

4. If the above problem (step 3) gives a column with a positive (resp. negative) reduced cost for a maximization (resp. minimization) problem, then update the master problem by adding the new column(s) and go to step 2. Otherwise, the current solution is optimal.

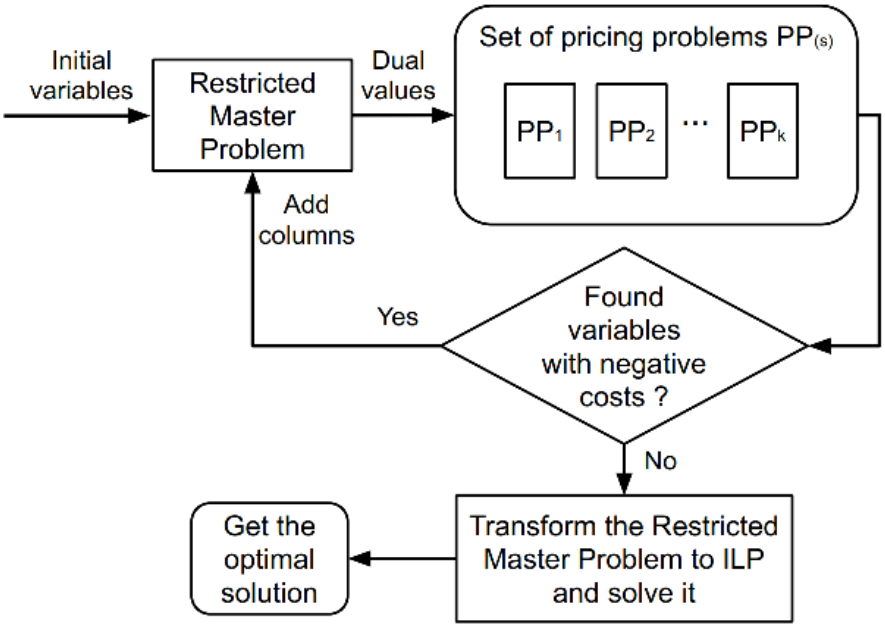


Figure 3.3: How column generation works

3.3.1 An example: path generation

The problem is as follows. Consider a network composed of a set η of nodes, a set ξ of links with given capacities $C_e(e \in \xi)$, and d, a connections (or a demand).

To connection d is assigned a predefined path set P_d between its end nodes s_d and t_d; any path p is identified with the set of links it traverses, i.e., $p \subseteq \xi$. Now let x_p denote the bandwidth allocated

to path $p \in P_d$. We are interested in finding a feasible flow maximizing $\sum_{p \in P_d} x_p$. In other words, this is the maximum flow problem formulated through path variables instead of flow variables on arcs. Obviously, these two formulations are equivalent if instead of a given set of paths P_d, all possible paths are included. Then, the idea is to solve the problem with a given set of paths P_d, and check its optimality by computing the reduced costs for all other paths not included in P_d. If there is some path with negative reduced cost, then it is added in P_d, and the problem is solved iteratively until the achieved solution reaches optimality. The above column generation algorithm is used here to build the candidate paths to join the set P_d. The entire formulation (using a restricted set of paths) follows:

$$max \sum_{p \in P_d} x_p \tag{3.1}$$

subject to :

$$\sum_{p \in P_d, e \in p} x_p \leq C_e \; ; \; \forall e \in \xi(\pi_e) \tag{3.2}$$

$$x_p \geq 0 \; ; \; p \in P_d \tag{3.3}$$

where π_e gives the dual variables. Note that with respect to an optimal basis B, the dual variables are expressed as $C_B B^1$ while the reduced cost is expressed as $C_B - C_B B^1 D$, where D collects the columns out of basis. In our formulation, we notice that C_B is composed of 1 for any path included in the basis (look at the objective function), and matrices B and D are composed of vectors (columns) of size $|E|$ having 1 at row e for each link e traversing the path. Hence, for a given path $q \in D$ the scalar product π_q gives $\sum_{e \in q} \pi_e$, which will be used to check the optimality of a given solution. As we are dealing with a maximization problem, an optimal basis should have the reduced costs all non positive, which comes to check if there exists a path q between s_d and t_d such that its reduced cost $1 - \sum_{e \in q} \pi_e$ is positive. Putting all this together, one needs to compute for some path q maximizing the reduced cost $1 - \sum_{e \in q} \pi_e$, which is equivalent to minimizing $\sum_{e \in q} \pi_e$. So, the problem of checking the feasibility of the solution provided by the master is reduced to the shortest path problem in the graph weighted with vector π and checking its corresponding reduced cost.

1. Similarly to the cutting stock problem, the algorithm may be presented as:

2. Initiate the master the problem with a set of some paths joining s_d to t_d.

3. Solve the master problem and note with the values of the obtained dual variables.

4. Compute the shortest path q in the graph weighted with (π) values.

5. If the reduced cost $1 - \sum_{e \in q} \pi_e \geq 0$, add q in the formulation and go to step 2; otherwise stop (the solution is optimal).

Chapter 4

An ILP formulation

Contenu

4.1 SYSTEM MODEL 58
4.1.1 Problem definition 58
4.1.2 Problem modeling 60
4.2 Problem statement and formulation 61
4.2.1 VNF reconfiguration problem: Problem statement 61
4.2.2 Problem formulation 61
4.3 Experimental results 65
4.3.1 Simulation Setup 65
4.3.2 Evaluation Metrics 66
4.3.3 Simulation Result and Analysis 67
4.4 Conclusion 70

In this chapter, We propose an ILP model for the VNF reconfiguration problem in the context of network slicing in 5G networks. The model takes into account two types of migrations (hot and cold migrations). Our algorithm generates a reconfiguration plan to pass rapidly and efficiently from an initial state where the placed VNFs are not optimally allocated to a new state, computed beforehand, respecting the resource capacity with minimal interruption, migration, and SLA costs. Section 6.1 presents the system model. Section 4.2 introduces the problem statement and formulation. The experiments and evaluation results are presented in Section5.3. We close the chapter with some concluding remarks.

4.1 SYSTEM MODEL

4.1.1 Problem definition

The problem we are interested in is the reconfiguration of 5G network slices. To define it simply, a reconfiguration is a reallocation of the NFV to adapt the utilization of network resources to the occurred changes. Fig. 4.1 shows an example of the VNFs reconfiguration problem. Three service slices are presented: self-driving car (slice uRLLC), live streaming (slice eMBB), and smart home (slice mMTC). Each slice is a set of virtualized network functions (VNFs), and each VNF is deployed in one Virtual Machine (VM) with different capacities (CPU, RAM). The slices (virtual resources) are deployed in the VNF infrastructure (physical resources) presented by five servers in the ”current state”, which represents the initial state of our problem. At time t, a new demand for slice deployment is presented. In this case, the current VNF placements become sub-optimal and inefficient. The VNFs placement should be reconfigured by migrating VNFs to another optimal state (target state). In our problem, the current and target states are known beforehand. Our objective is to reconfigure all the realized migrations to attain the target state (new placement of VNFs) and generate a reconfiguration plan that allows passing rapidly and optimally from the current state to the target state while respecting resource capacities, minimizing the service interruption and migration duration.

The VNF migrations are performed in two types of migrations. In a **hot (live) migration**, the running VNFs are moved between the source and target servers without disconnecting the service or application. The hot migration ensures minimal downtime for the VNF. In a **cold (non-live) migration**, the VNFs are moved between servers by powering off the VNF on the source server, moving it to the

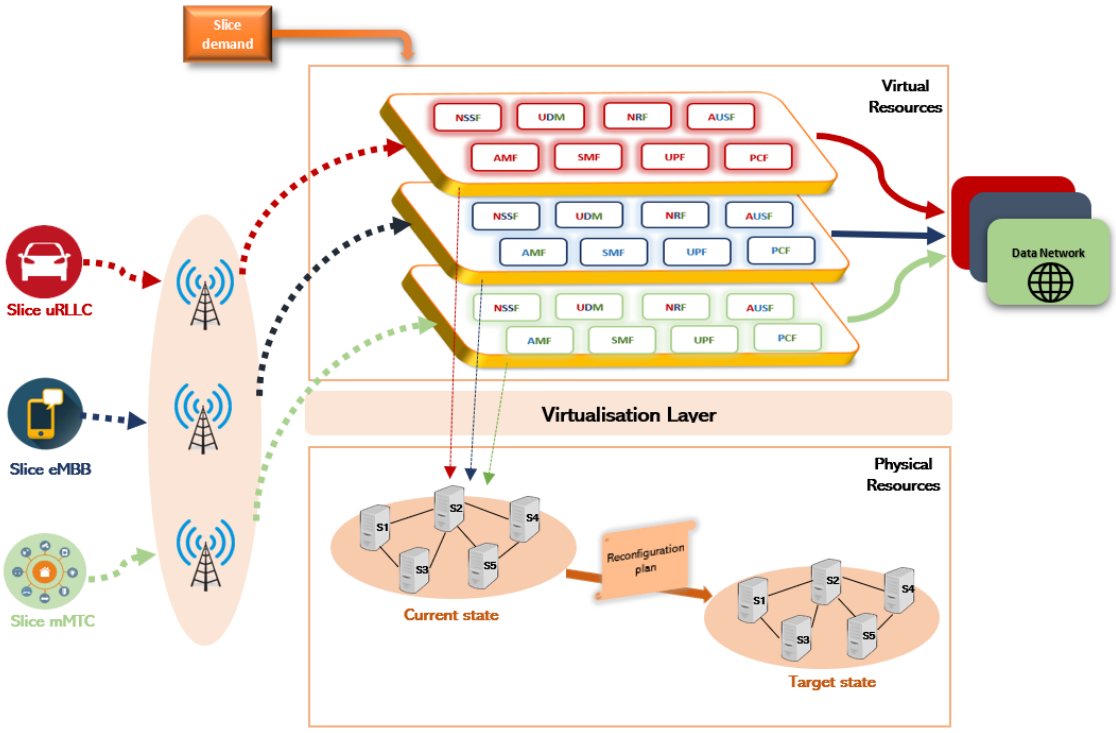


Figure 4.1: Presentation of VNFs reconfiguration problem. **NSSF**: *Network Slice Selection Function*, **UDM**: *Unified Data Management*, **NRF**: *Network Repository Function*, **AUSF**: *Authentication Server Function*, **AMF**: *Access and Mobility Management Function*, **SMF**: *Session Management Function*, **PCF**: *Policy Control Function*

target server then powering it back up on the target server. The cold migration ensures an important downtime that should be minimized.

Our objectives are to minimize the total migration duration so that the VNF migration be performed as quickly as possible and to minimize the service interruption, which is important, especially for some use cases such as smart grids, intelligent transport systems, and remote surgery. These services require ultra-high network reliability of more than 99.999% and very low latency (of 1 millisecond) for packet transmission. Minimizing the service interruption ensures the slice availability and the avoidance of SLA violations. Moreover, the VNF interruption degrades the service not only for one slice but for many other slices too as they can be shared between several of them.

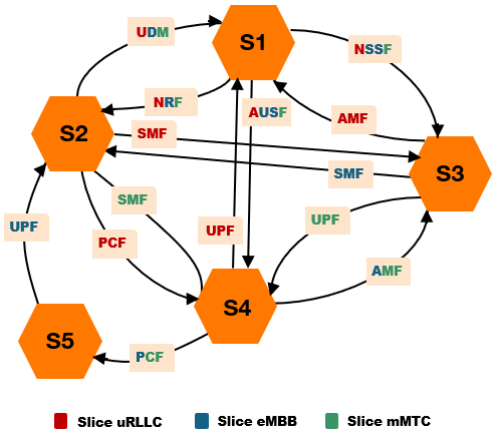


Figure 4.2: The graph representation of VNFs migrations

4.1.2 Problem modeling

The VNF reconfiguration problem is a NP-Hard optimization problem. [29] demonstrates the NP-hardness of the reconfiguration problem in the context of distributed systems. In this paper, we model the network as a connected directed graph $G = (V, E)$. The nodes $v \in V$ stand for the servers and the links $(v, v') \in E$ connecting the nodes represent the VNF migration from node v to node v' (see Fig. 4.2). The VNF can be shared between different slices (ex in Fig. 4.1: *NSSF*, *UDM*, *NRF* and *AUSF* are shared VNFs between all slices) or dedicated to one slice (ex in Fig. 4.1: *SMF*). In this paper, we assume that each VNF is implemented in one VM and there is a low communication delay between servers. Thus, the flow routing is not taken into consideration, nor are the precedence constraints.

Our objective is to propose an optimization algorithm that takes as input the current and target states and generates as output the reconfiguration plan while minimizing the total migration duration and the service interruption. There is a property that already has been demonstrated in [29], and consists of finding the reconfiguration plan in polynomial time without service interruption using the Topological Sorting (TS) algorithm, in the case where the network topology is an acyclic graph.

The topological sorting for Directed Acyclic Graph (DAG) [69] [70] is a linear ordering of nodes such that for every directed arc (v, v'), node v comes before v' in the ordering. The TS algorithm is not possible if the graph is not a DAG (for the case of a cyclic graph). For this reason, we propose an exact model that can be applied to different types of graphs (acyclic and cyclic) and where the

solution can be optimized in terms of service interruption and total migration duration.

4.2 Problem statement and formulation

Linear programming constitutes the basis of the solution method developed in this work. In this section, we present the VNF reconfiguration problem statement and its formulation as an Integer Linear Programming (ILP).

4.2.1 VNF reconfiguration problem: Problem statement

The VNF reconfiguration problem is presented as follows with notation given in Table 4.1 for easy reference:

- ***Given***: The placement of VNFs in the current and target states.
- ***Find***: in which stage k the VNF_i should be migrated, which type of migration to use (cold or live migration) while respecting the resource constraints.

  The total number of stages N required for all VNFs to be migrated can be equal to N_v the number of VNFs in worst cases, where each VNF migrates separately in one stage. Or less than that, in cases where we have parallel migrations.
- ***Subject to***: the VNF occupied CPU capacity cap_i^{cpu}, the VNF occupied RAM capacity cap_i^{ram}, the VNF interruption duration δ_i and the VNF migration duration T.
- ***Objective***: minimizing the VNF migration and interruption duration.

4.2.2 Problem formulation

To formulate the integer linear programming model, we introduce the decision variables, the constraints to be satisfied, and the objective function.

4.2.2.1 Decision variables

We have the following decision variables to model VNF migrations between servers (Knapsacks):

$$x_{ik} = \begin{cases} 1, & \text{if the } VNF_i \text{ is migrated in stage } k \text{ ;} \\ 0, & \text{otherwise.} \end{cases} \tag{4.1}$$

Table 4.1: Table of Notations

Notation	Description
N	number of stages
N_v	number of VNFs
N_s	number of servers
k	order / stage of the reconfiguration
x_{ik}	a binary variable indicating that VNF_i is migrated in order k
y_{ik}	a binary variable indicating the stage where VNF_i is interrupted in source host
$O_{(s)}$	set of VNFs originating from server s
$D_{(s)}$	set of VNFs targeting server s
C_s^k	represents the residual CPU capacity of server s in stage k
R_s^k	represents the residual RAM capacity of server s in stage k
cap_i^{cpu}	represents the occupied CPU capacity of VNF_i
cap_i^{ram}	represents the occupied RAM capacity of VNF_i
δ_i	represents the interruption duration of VNF_i
T	represents the migration duration of a given VNF
β_i	represents the cost of service interruption, which is the SLA availability of each VNF_i
α	represents the migration cost of all VNFs

$$y_{ik} = \begin{cases} 1, & \text{if the } VNF_i \text{ is interrupted in stage } k \text{ ;} \\ 0, & \text{otherwise.} \end{cases} \tag{4.2}$$

4.2.2.2 Problem constraints

- *Integrity constraint for migration*

Equation (4.3) insures that VNF_i can only be migrated once to the destination server.

$$\sum_{k=1}^{N_v} x_{ik} = 1 \ ; \ \forall i \in \{1, .., N\} \tag{4.3}$$

- *Integrity constraint for interruption*

Equation (4.4) shows that VNF_i can only be interrupted once in the source server.

$$\sum_{k=1}^{N_v} y_{ik} = 1 \ ; \ \forall i \in \{1, .., N\} \tag{4.4}$$

- *Capacity constraint*

Equations (4.5) and (4.6) ensure that resource capacities of each server (for CPU and RAM, respectively) in each stage k, are not exceeded. VNFs that are interrupted free the resources of their origin server, while VNFs that are placed consume the resources of their destination server.

$\forall s \in \{1, .., N_s\} \ ; \ \forall k \in \{1, .., N\} \ ;$

$$C_s^k - \sum_{i \epsilon D(s)} x_{ik} cap_i^{cpu} + \sum_{i \epsilon O(s)} y_{ik} cap_i^{cpu} = C_s^{k+1} \tag{4.5}$$

$\forall s \in \{1, .., N_s\} \ ; \ \forall k \in \{1, .., N\} \ ;$

$$R_s^k - \sum_{i \epsilon D(s)} x_{ik} cap_i^{ram} + \sum_{i \epsilon O(s)} y_{ik} cap_i^{ram} = R_s^{k+1} \tag{4.6}$$

$$C_s^k \geq 0 \ ; \ \forall s \in \{1, .., N_s\} \ ; \ \forall k \in \{1, .., N\} \tag{4.7}$$

$$R_s^k \geq 0 \ ; \ \forall s \in \{1, .., N_s\} \ ; \ \forall k \in \{1, .., N\} \tag{4.8}$$

- *Interruption duration constraint*

Equation (4.9a) ensures that the VNF is migrated with cold migration during the whole process. It considers that interruption and migration could be performed in the same stage. Otherwise, equation (4.9b), which refers also to cold migration, gives more flexibility. The interruption and migration could be performed in one or more stages. In cold migration, the VNF is interrupted first in the source host then it is migrated to the destination host. The VNF interruption should be before VNF migration.

Equation (4.9c) ensures that the VNF is migrated with live migration during the whole process. In this case, the migration of VNF_i occurs at one stage before the interruption. Here, we consider that VNF is interrupted in the source host after totally being migrated to the destination host.

Equation (4.9d) encompasses the cold and live migration.

$$\sum_{k=1}^{N_v} ky_{ik} = \sum_{k=1}^{N_v} kx_{ik} \; ; \; \forall i \in \{1, .., N\} \tag{4.9a}$$

$$\sum_{k=1}^{N_v} ky_{ik} \leq \sum_{k=1}^{N_v} kx_{ik} \; ; \; \forall i \in \{1, .., N\} \tag{4.9b}$$

$$\sum_{k=1}^{N_v} ky_{ik} = \sum_{k=1}^{N_v} kx_{ik} + 1 \; ; \; \forall i \in \{1, .., N\} \tag{4.9c}$$

$$\sum_{k=1}^{N_v} ky_{ik} \leq \sum_{k=1}^{N_v} kx_{ik} + 1 \; ; \; \forall i \in \{1, .., N\} \tag{4.9d}$$

The interruption time is considered as the number of stages between the VNF interruption and VNF migration. In live migration, the interruption time is negligible, therefore, we consider $\delta_i = 0$. In cold migration, the VNF interruption is performed at least in one stage $\delta_i = 1$. Equation (4.10) refers to the formulation of VNF interruption time.

$$\delta_i = (\sum_{k=1}^{N_v} kx_{ik} + 1) - (\sum_{k=1}^{N} ky_{ik}) \tag{4.10}$$

- *Migration duration constraint*

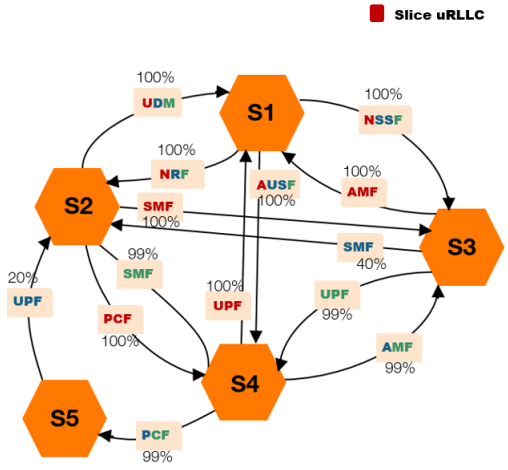
Equation (4.11) finds the maximum migration duration that should be minimized.

$$\sum_{k=1}^{N_v} kx_{ik} \leq T \; ; \; \forall i \in \{1, .., N\} \tag{4.11}$$

4.2.2.3 Objective function

$$min(\sum_{i=1}^{N_v} \beta_i \delta_i + \alpha T) \tag{4.12}$$

The objective function consists of minimizing the VNFs interruption time, which represents the number of stages during which the VNF_i is interrupted and minimizing the VNFs migration duration, which represents the number of stages during which the VNF_i is migrated. The weight β_i associated



(a). The VNFs migration with the SLA availability according to each service slice.

Start Step	Actions	Is interrupted ?	Interruption duration
1	Migrate NSSF to S3	NO	--
1	Migrate AUSF to S4	NO	--
1	Migrate SMF to S3	NO	--
1	Migrate UPF to S1	NO	--
1	Migrate PCF to S5	NO	--
2	Migrate NRF to S2	NO	--
2	Migrate UPF to S4	NO	--
2	Migrate UDM to S1	NO	--
2	Migrate PCF to S4	NO	--
3	Migrate AMF to S1	NO	--
3	Migrate SMF to S2	YES	3
3	Migrate AMF to S3	YES	1
3	Migrate UPF to S2	YES	3

(b). The reconfiguration plan for all slices.

Figure 4.3: The reconfiguration plan of all VNFs migrations taking into consideration the SLA availability

with δ_i represents the SLA availability for each VNF belonging to a given slice. The service availability of the slice is divided into three ranges: high availability ($\beta_i = 100\%$), average availability ($\beta_i \geq 99\%$), and low availability ($\beta_i < 99\%$).

4.3 Experimental results

4.3.1 Simulation Setup

4.3.1.1 Topology Dataset

The ILP model is solved using CPLEX Optimisation studio V12.8 integrated with python. Experiments were conducted on a machine with a Core i7-6600U CPU and 16 Go of RAM. We use randomly generated topologies to evaluate our model for both acyclic and cyclic graphs. The graphs are randomly generated with different sizes (small and medium graphs) using the NetworkX, which is a well-known python lib, and we are inspired by the code proposed by [71] [72]. The nodes of the graph represent the servers and the links represent the VNF migration. The node and link capacities are generated randomly, (1∼50) for CPU capacity and (10∼90) for RAM capacity.

4.3.1.2 VNF and slice Datasets

The type number of VNFs is randomly generated in the range (20∼150). The slices are randomly generated by choosing the set of connected VNFs, taking into consideration the shared and dedicated VNFs. Each slice contains at least 5 VNFs. The datasets are presented in Table 5.1 and Table 5.2.

Table 4.2: Datasets of acyclic graphs

Instances	Servers	VNFs	Slices
DC-acy1	10	25	6
DC-acy2	20	35	11
DC-acy3	40	60	12
DC-acy4	50	120	24
DC-acy5	80	150	35

Table 4.3: Datasets of cyclic graphs

Instances	Servers	VNFs	Slices
DC-cy1	10	30	8
DC-cy2	20	45	12
DC-cy3	40	70	15
DC-cy4	50	120	25
DC-cy5	80	146	32

4.3.2 Evaluation Metrics

To show the performance of our model, we use the following evaluation metrics:

- **Scalability**: To evaluate the scalability of our model, we adopt two metrics. These metrics are the model execution time in seconds and the estimated gap to optimal in % after one hour.

- **Migration duration**: We evaluate the total migration duration for the entire process, as well as the number and the percentage of migrated VNFs per each step of the migration.

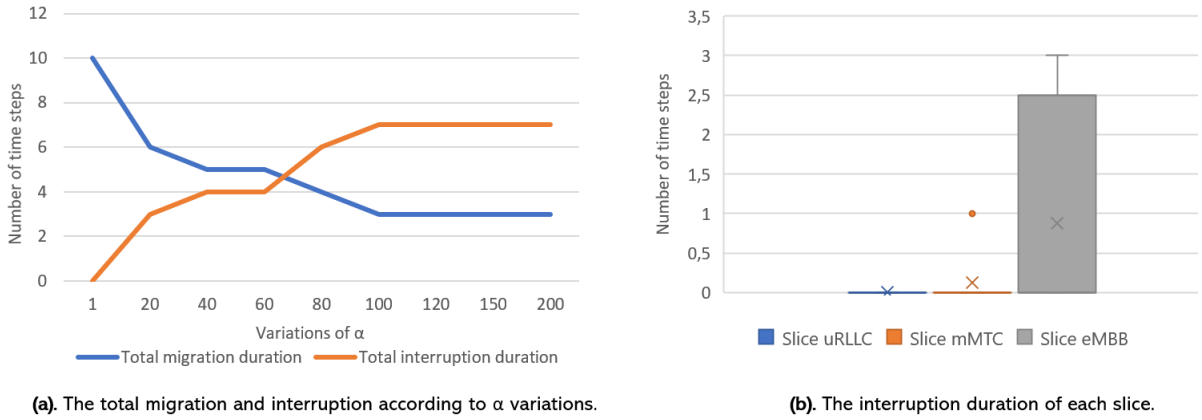
- **Interruption duration**: We evaluate the ratio of interrupted VNFs to the total VNFs as well as the interruption duration for each slice demand.

- **Migration and interruption costs**: We evaluate the interruption and migration cost by giving each VNF the corresponding SLA availability β_i and varying the weight α.

4.3.3 Simulation Result and Analysis

4.3.3.1 Evaluation according to the nature of slices

We evaluate the example presented in Fig. 4.1 with $V = 5$ and $E = 14$. As we mentioned earlier, each slice has its SLA availability that should be respected. In Fig. 4.3(a), we present the considered values of each service availability: high availability for slice uRLLC ($\beta_i = 100\%$), average availability for slice mMTC ($\beta_i = 99\%$) and low availability for slice eMBB ($\beta_i < 99\%$).



(a). The total migration and interruption according to α variations.

(b). The interruption duration of each slice.

Figure 4.4: The evaluation results of migration and interruption cost

Fig. 4.4(a) shows the total migration and interruption duration for all slices according to different variations of α, where the number of α is varied between (1∼200). We can see that the total migration duration decreases and the total interruption duration increase with the rise of the α. From $\alpha = 100$, we can observe clearly that the total migration and interruption duration stagnates respectively in steps 3 and 7. This is because the ILP model finds the optimal solution that minimizes both migration and interruption duration.

To evaluate the interruption duration for each slice, we set the α to 100. Fig. 4.4(b) presents the migration duration of VNFs for each slice. We can see that in the slice uRLLC there is no interruption as it demands high availability, then for mMTC, there is one VNF interrupted for a duration of 1 step, while the eMBB has more interrupted VNFs. To have more details about the interrupted VNFs, Fig. 4.3(b) shows the reconfiguration plan of VNFs migrations. We can see that UPF and SMF dedicated to eMBB are interrupted for 3 steps. This is because of the low SLA availability of 20% and 40% respectively. Then, the AMF shared between eMBB and mMTC is interrupted for 1 step,

which explains the importance of the service availability. The ILP model takes into consideration the availability of each slice while minimizing the interruption duration.

4.3.3.2 Evaluation according to the nature of datasets

In this section, we evaluate the datasets presented in Table 5.1 and Table 5.2 for acyclic and cyclic graphs, respectively. In this experiment, we set β_i and α to 1 to focus more on the nature of graphs and their impact on the results.

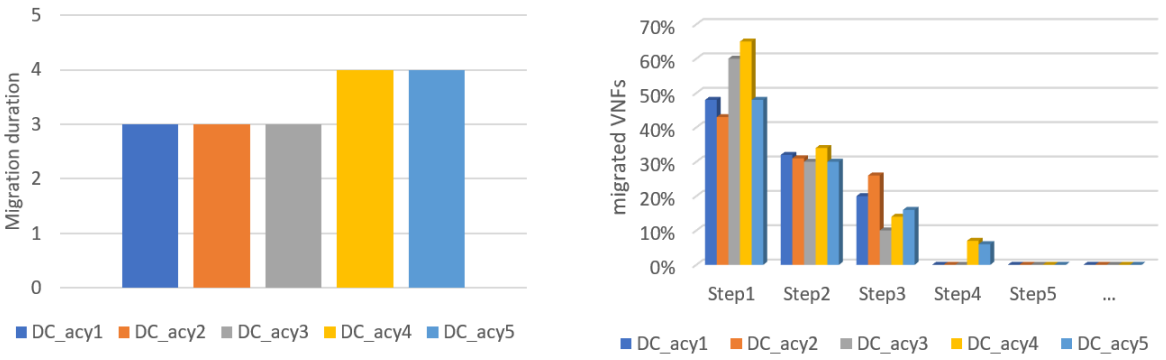
- **Acyclic graph**

For acyclic graphs, our ILP model solves the VNF reconfiguration problem without interruption and with 0% of optimality gaps. As we mentioned in Section 4.1.2, in the case of an acyclic graph, we can find the reconfiguration plan in polynomial time without interruption using the TS algorithm. In Table 4.4, we compare our ILP model with the TS algorithm. We can see that the TS finds a reconfiguration plan in milliseconds compared to our ILP model. However, the best objective of our ILP model is more interesting than the TS algorithm. This is because the ILP finds the optimal solution that minimizes the migration duration while the TS finds a feasible solution without taking into consideration the migration duration. This means that the ILP gives a solution where the VNFs are migrated from the early steps and in parallel as long as possible (see Fig. 4.5(b)) and with minimum migration duration (see Fig. 4.5(a)). Fig. 4.5 shows that the ILP migrates all VNFs in the first four steps for all instances.

Table 4.4: Comparison between the ILP model and TS algorithm for acyclic graph

Instances	**ILP**: Execution time (s)	**ILP**: Best objective	**TS**: Execution time (s)	**TS**: Best objective
DC-acy1	0.33	3	0.000532	25
DC-acy2	1.06	3	0.000324	35
DC-acy3	2.08	3	0.000949	60
DC-acy4	17.76	4	0.001346	80
DC-acy5	36.19	4	0.001257	150

- **Cyclic graph**

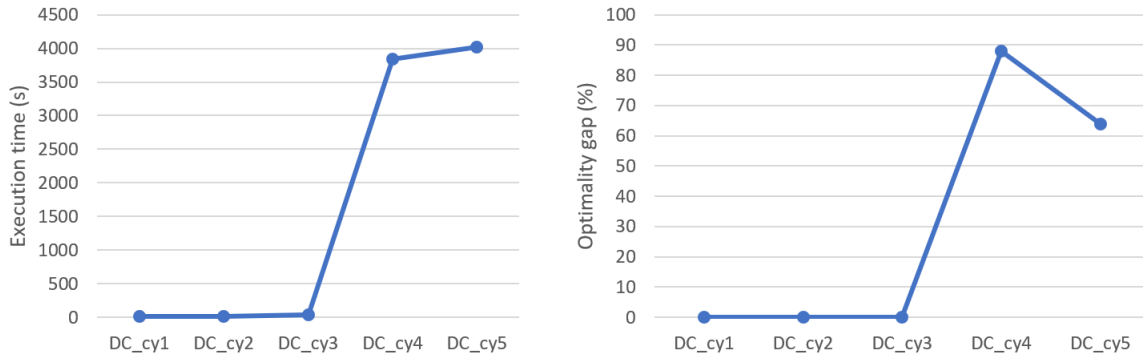


(a). Total migration duration.

(b). Percentage of migrated VNFs per steps of migrations.

Figure 4.5: The evaluation results of migration duration for the acyclic graph

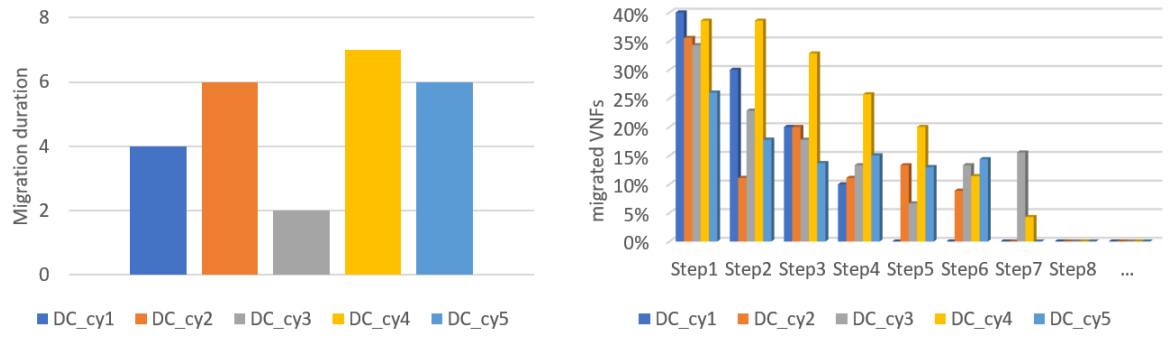
Figures 4.6(a) and 4.6(b), respectively, show the evolution of the execution time and the gap to optimal estimated by CPLEX at the end of the execution time according to the different instances. These two metrics significantly increase for DC-cy4 and DC-cy5 instances due to the np-hardness of the problem. Optimality gaps are often at 0% except in the case of DC-cy4 and DC-cy5 instances which need more time to find an optimal solution. The ILP converged to optimality for medium graphs (120 to 146 VNFs) about over an hour of simulation. It provides an interesting solution in terms of migration duration and VNF interruption. Fig. 4.7(a) shows that the VNFs can migrate over 7 steps for DC-cy4 and over 6 steps for DC-cy5. The interrupted VNFs are less than 20% and 10%, respectively, for DC-cy4 and DC-cy5 (see Fig. 4.7(a)). In Fig. 4.7(b), we can see that most VNFs are migrated without interruption (hot migration where $\delta_i = 0$).



(a). Execution time variations.

(b). Optimality gap variations.

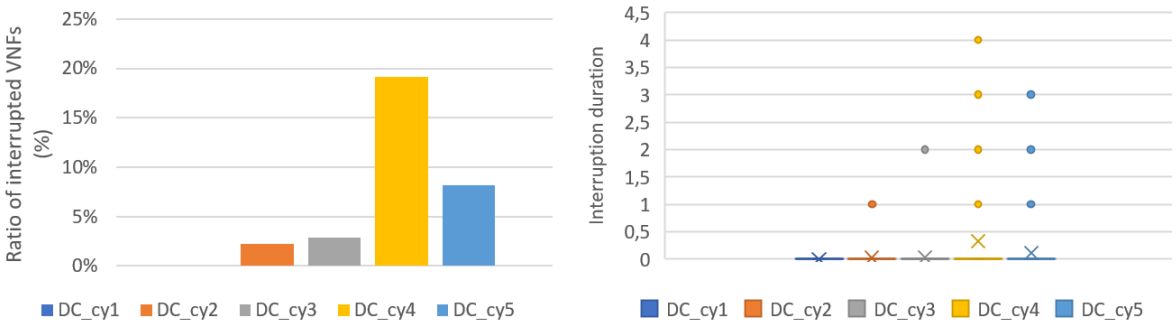
Figure 4.6: Scalability evaluation results for cyclic graph



(a). Total migration duration.

(b). Percentage of migrated VNFs per steps of migrations.

Figure 4.7: The evaluation results of migration duration for cyclic graph



(a). Ratio of interrupted VNFs.

(b). Interruption duration for each instance.

Figure 4.8: The evaluation results of interruption duration for cyclic graph

Like acyclic graphs, the ILP succeeds for cyclic graphs to migrate efficiently and quickly the VNFs from the first steps with minimum interruptions. However, in acyclic graphs, the migrations are performed without interruptions and are slightly faster when compared to the case cyclic graphs. This leads to conclude that the ILP model complexity depends strongly on the presence of cycles.

4.4 Conclusion

In this chapter, we have proposed an ILP-based solution for the problem of slice reconfiguration in the context of 5G networks. The ILP finds a reconfiguration plan, consisting of a series of migrations that will relocate the VNFs from their current servers to those computed beforehand while minimizing the migration and interruption duration. We evaluate the proposed model according to the service

importance taking into consideration the SLA availability metric, and according to the nature of the datasets (whether it is an acyclic or a cyclic graph). The evaluation results show that the ILP finds the optimal reconfiguration plan that minimizes the service interruption, the total migration duration, and service availability. In the next chapter, we propose a heuristic algorithm in order to improve the CPU time and allow dealing with larger instances.

Chapter 5

A heuristic approach

Contenu

5.1 The problem modeling 74
5.2 Related work 76
5.3 Overview of the proposed approach 77
5.3.1 Heuristic used for the Feedback Arc Set problem 77
5.3.2 Parallel Topological Sorting algorithm 78
5.4 Experimental results 80
5.4.1 Evaluation Metrics 80
5.4.2 Simulation Result and Analysis 81
5.5 Conclusion 84

In this chapter, we propose a new heuristic for the VNF reconfiguration problem in the context of 5G network slicing. The Feedback Arc Set (FAS) problem is used and an efficient heuristic inspired by the concept of the Topological Sorting (TS) algorithm is proposed. This approach generates a reconfiguration plan to pass rapidly from an initial state where the placed VNFs are not optimally allocated to a new state, computed beforehand while minimizing the VNF interruption and migration duration. Our findings are illustrated by numerical results which are compared to these of the ILP model already reported in Chapter 4. Another issue dealt with in this work is the overall duration of the migration process. We are interested in scheduling as much as possible the migration process by realizing migrations in parallel. All this justifies calling for parallel topological sorting.

5.1 The problem modeling

In this work, a graph model approach is used in order to find a heuristic for the 5G reconfiguration problem. The problem is modeled using graph theory. $G = (V, E)$ is a directed graph where $V = 1, 2, ..n$ stands for the servers (n is the total number of servers). Initially each server i has a remaining resource (i.e. residual capacity, $C_i \geq 0$ and residual memory $R_i \geq 0$). E represents a set of edges, $\forall (u, v) \in E$, u represents the server at the initial state on which the VNF is executed and v represents the server at the final state on which the VNF need to be migrated to.

We recall equations (5.1) and (5.2) that should be verified in the final state. The equations ensure that for a given vertex v of G once we migrate all the VNFs running on a given server, it will have enough resources to process the entering VNFs. In this article, we consider servers with two resources type (CPU and ram) as noted in equations (5.1) and (5.2).

$$\forall v \in V, \quad C_v + \sum_{t \in NE_G[v]} cpu_{vt} \geq \sum_{s \in NI_G[v]} cpu_{sv} \tag{5.1}$$

$$\forall v \in V, \quad R_v + \sum_{t \in NE_G[v]} ram_{vt} \geq \sum_{s \in NI_G[v]} ram_{sv} \tag{5.2}$$

Where:

- $NE_G[v]$ design the set of *out-neighbors* vertices in G of v, that is $\forall t \in NE_G[v]$, $(v, t) \in E$.
- $NI_G[v]$ design the set of *in-neighbors* vertices in G of v, that is $\forall s \in NI_G[v]$, $(s, v) \in E$.

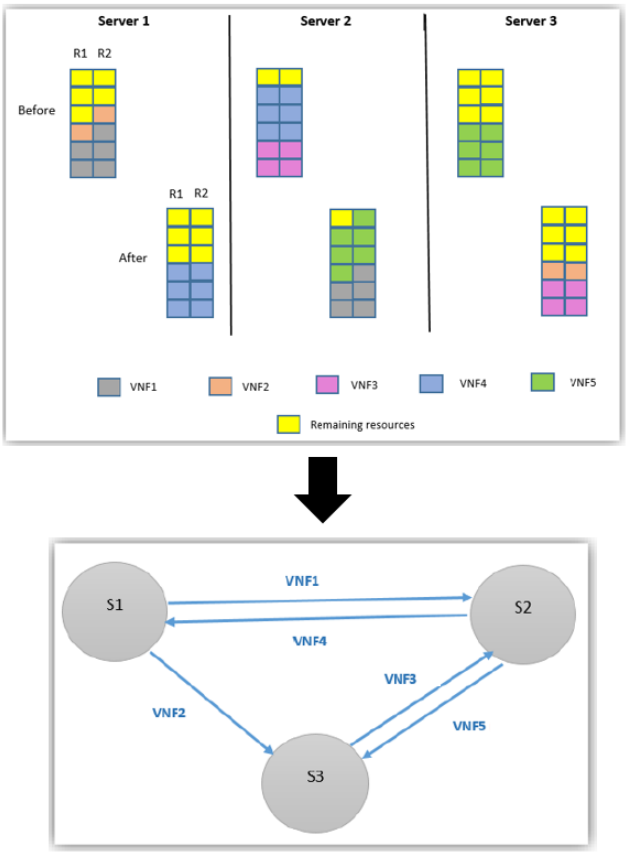


Figure 5.1: Graph model

Fig. 5.1 provides an example to show how to model the problem using graph theory. At the top of Fig. 5.1 we can see three servers, each one has represented as a box for the initial state 'Before' and the final state 'After'. Here it's assumed that every server has 2 resources, with R_1 representing the capacity resource (the left column of the box) and R_2 the memory (the right column of the box). The different colors represent different VNFs running on the servers, for example in server 1 there are two VNFs. VNF1 uses 2 resource units R_1 and 3 units of R_2 resource, while VNF2 uses 1 unit of R_1 resource and one unit of R_2. The yellow color highlights the remaining resources in a server. At the bottom of Figure 5.1, we find the graph model. The vertices represent the servers and the edges represent the VNF migrations. Our objective is to reconfigure the VNF migrations which are represented by edges in order to find the reconfiguration plan.

The graph representation could be acyclic or cyclic according to the network complexity. In the case of an acyclic graph, we know from [29], that we may build a reconfiguration plan in polynomial

time without service interruption using the TS algorithm. The topological sorting for Directed Acyclic Graph (DAG) [69] [70], is a linear ordering of nodes such that for every directed arc (u, v), node u comes before v in the ordering. The nodes in a directed graph can be arranged in a topological order if and only if the directed graph is acyclic [61]. In this article, we propose a heuristic inspired by the algorithm of topological sorting that can be applied to different types of graphs (cyclic and acyclic).

5.2 Related work

In the literature, the topological sorting algorithm has been applied in different domains. But, up to our knowledge, it is not yet studied for the VNF reconfiguration problem.[73], [74], [75], [70] and [59] treat the topological sorting (or topological ordering) with large graphs. In [73] and [74], a new topological sorting algorithm was proposed for large graphs that decrease the computing time and increase the efficiency, better than Kahn's and Depth First Search (DFS) topological sorting algorithms. [75][70][59] The Kahn's and DFS algorithms are the basic algorithms of graph theory for topological sorting. Kahn's algorithm is described in [59]. It works by keeping track of the number of incoming edges into each node (indegree) and repeatedly stores the nodes with zero in-degree and deletes the edges originating from them until no element with zero in-degree can be found. On the other hand, the DFS algorithm traverses the graph as long as possible (i. e., until there are no more non-visited successors left) along a branch before tracking back. In [75], the DFS was used to reduce the temporary space of a large Directed Acyclic Graph (DAG). The authors present the depth-first discovery algorithm (DFDA) to find topological sorting. [70] presents an I/O efficient algorithm called IterTS for topologically sorting directed acyclic graphs. However, the algorithm is inefficient in the worst case.

Topological sorting in parallel is presented in [76] and [77]. In [76], the authors proposed a new topological sorting algorithm based on the parallel computation approach. The idea of the parallel computation in this article consists in traversing, in parallel, all links leading from a node, once this node is visited. The algorithm was implemented using a Stream-Multiple Data stream (SIMD) machine. The time complexity of this proposed algorithm is the order of the longest distance between a source node and sink node in an acyclic digraph. Besides, it [77] is presented the Parallel Genetic Algorithm (PGA) using topological sorting. The algorithm improves the solution of the Job Shop Scheduling Problem (JSSP) and minimizes the execution time of the makespan calculation. To cal-

culate the makespan of each schedule, the authors used the longest path algorithm. The topological sorting is found by calculating the degree for all the vertices and starting from the vertex which has a degree of 0. The sequences of each schedule are executed in parallel. In this work, we provide a new algorithm framework applied to the problem of VNF reconfiguration called Parallel Topological Sorting (PTS) that minimizes the migration duration of VNFs.

5.3 Overview of the proposed approach

The approach is starting with the pre-processing phase where we model the migration of concerned VNFs. If the resulting graph is cyclic, which is often the case in 5G networks, we find an acyclic sub-graph using the Feedback Arc Set (FAS: a set of edges that when removed from a cyclic graph G give an acyclic graph G'). Once an acyclic graph is obtained, we use the parallel topological sorting algorithm while migrating all possible running or interrupted VNFs (edges removed from the cyclic graph after applying the FAS algorithm). The reconfiguration plan is built by putting together all VNFs migrations (see Fig. 5.2).

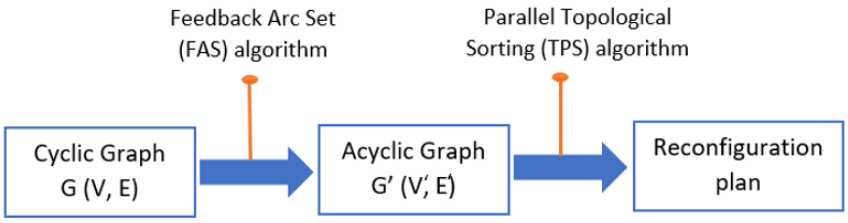


Figure 5.2: The proposed approach for cyclic graph

5.3.1 Heuristic used for the Feedback Arc Set problem

The problem is solved by a greedy heuristic for the minimum set cover problem. It consists in gradually building the feedback arc set by always picking an arc as the next element that, when removed, breaks most of the remaining simple cycles. This heuristic of FAS used in our approach is the one used in [61].

Mathematically, the FAS problem is formulated as follow:

$$\min_{y} \sum_{j=1}^{m} w_j y_j \tag{5.3}$$

subject to:

$$\left\{ \begin{array}{l} \sum_{j=1}^{m} a_{ij} y_j \geq 1 \quad for \quad each \quad i = 1, 2, ..., l \\ y = \{0, 1\} \end{array} \right\}$$

Where, m denotes the number of edges; w_j are non-negative weights (often integer); y_j takes 1 if edge j is in the feedback edge set, and 0 otherwise; a_{ij} is 1 if edge j traverses cycle i, and 0 otherwise; l denotes the number of simple cycles. The matrix $A = (a_{ij})$ is called the cycle matrix.
The weakness of this formulation is that enumerating all simple cycles may be intractable. So the real challenge stands in finding an intelligent enumeration of all simple cycles. This leads to enumerating simple cycles in a lazy fashion and extends an incomplete cycle matrix iteratively until a minimum feedback arc set is found (note that the minimum feedback arc set problem is an NP-hard problem on graphs that seeks a minimum set of arcs which leaves a directed graph G acyclic when removed [61]). We provide an overview of the FAS algorithm in Fig. 5.3, for more details we reference[61].

5.3.2 Parallel Topological Sorting algorithm

In our algorithm, we take into consideration the residual resources for nodes, Equations (5.1) and (5.2) are verified at every step of the algorithm.

In the case of an acyclic graph, we start by finding a list of sink nodes that have no outcoming arcs and insert them into a set of the sinks called S. Instead of looking for one sink node to start by (as the Kahn's algorithm [59]), we search for a set of all sink node S. Then we construct a subgraph from the original graph where we remove the nodes of S and the edges entering to the nodes stored in S, then add these arcs to the ordering. We decrement the out-degree of nodes originating from deleted edges. We iterate this process until reaching an empty subgraph. The termination of the algorithm is assured by the fact that at every step we'll remove at least one sink node (the subgraphs are acyclic as we start with an acyclic graph, so they have at least one node with out-degree 0) and we have a finite number of nodes.

The novelty of our algorithm lies in the removed arcs of the feedback arc set F. In this case, after finding the sink nodes S, we realize all incoming migrations to S if the residual capacity of S is sufficient to accommodate all incoming VNFs. Otherwise, we interrupt FAS edges outcoming from S to leave enough space in order to be able to migrate the VNFs. The migrated VNFs are added to the ordering, and the capacity and out-degree are updated. In order to minimize the migration duration,

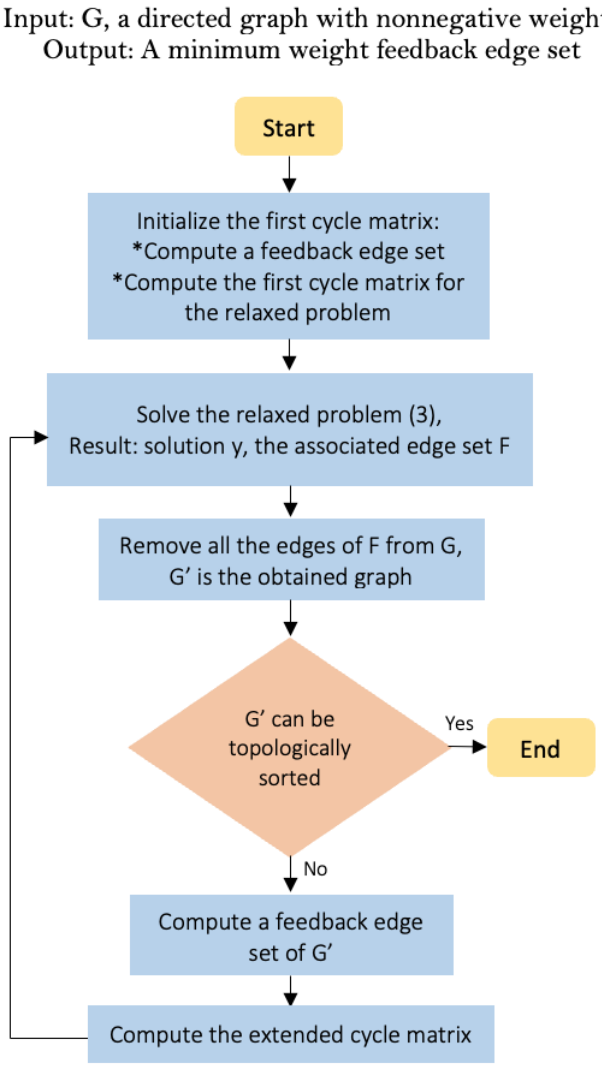


Figure 5.3: Overview of the FAS algorithm

we migrate all possible VNFs of the feedback arc set while checking if there is enough place for FAS migrations. We check if the resource capacity of edges in F entering removed nodes is less than the resource capacity of removed nodes. In this case, we can add the edges to the ordering called the reconfiguration plan RF. We iterate this process until there are no nodes left with zero out-degree. The remaining edges of F not inserted during the process are added to the end of the ordering (see Fig. 5.4). We provide a pseudo-code (algorithm 1) for the PTS algorithm and an example of all the process (Fig. 5.8) for better understanding.

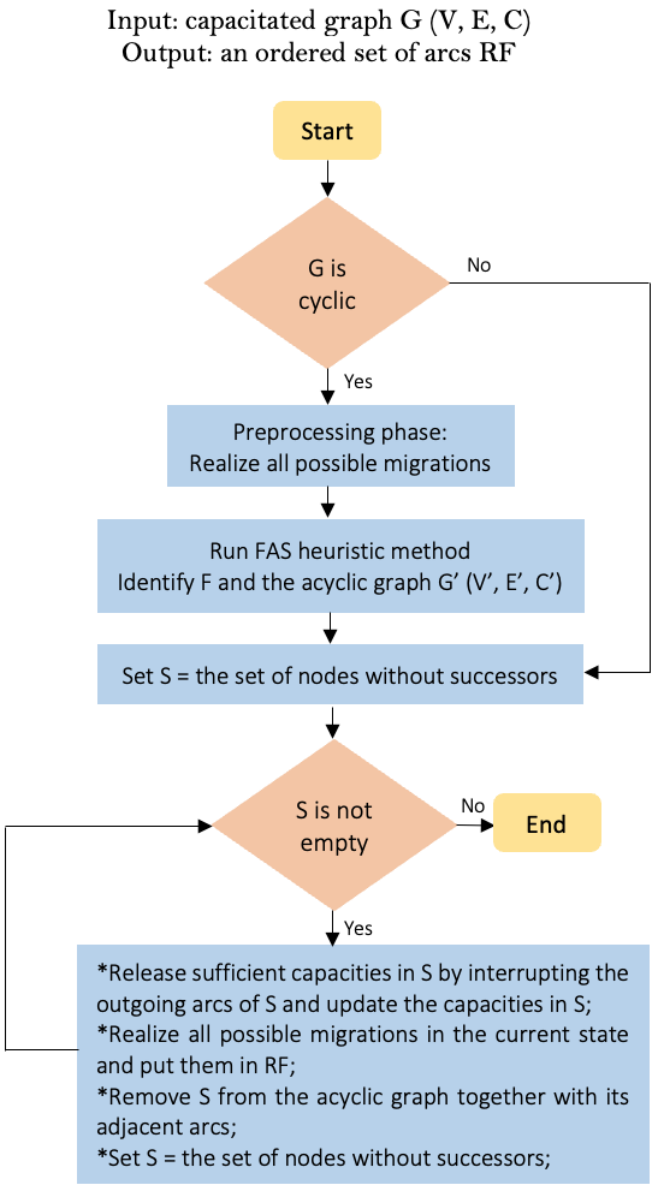


Figure 5.4: Diagram of the proposed approach

5.4 Experimental results

5.4.1 Evaluation Metrics

To show the performance of our algorithm, we use the following evaluation metrics:

- **Scalability**: We evaluate the execution time in seconds.

Algorithm 1: Parallel topological sorting algorithm for the cyclic graph

Data: A cyclic graph $G' = (V', E', C')$, An acyclic graph $G = (V, E, C)$ and F a feedback arc set
Result: Reconfiguration plan (RF) for all VNFs (E' edges of G')

while $G \neq \emptyset$ **do**
  Set $S = \{v \in V : D_+(v) = \emptyset\}$;
  Set $I(v)$ = in-edges to v in G ;
  Set $OutF(v) = \{C_{E'}, E' \in F\}$ out-edges from v ;
  Set $InF(v) = \{C_{E'}, E' \in F\}$ in-edges to v ;
  if $C_v \geq \sum_{E \in I} C_E$ **then**
    Add E to RF ;
    Set G = subgraph of G with $V \setminus \{S\}$;
    Update D_+ and C ;
  else
    while $C_v \leq \sum_{E \in I} C_E$ **do**
      Interrupt $E' \in F$;
      $C_v \leftarrow C_v + C_{E'}$, $C_{E'} \in OutF(v)$;
      if $C_v \geq \sum_{E \in I} C_E$ **then**
        Add E to RF ;
        Set G = subgraph of G with $V \setminus \{S\}$;
        Update D_+ and C ;
      else
    end
  end
  for E' *in* F **do**
    if $C_{E' \in InF(v)} \leq C_v$ **then**
      Add E' to RF;
    else
      Add E' to the end of RF;
    end
  end
end

- **Migration duration**: We evaluate the total migration duration for all the process.

- **Interruption duration**: We evaluate the total interruption duration for all the process.

5.4.2 Simulation Result and Analysis

In this section, we evaluate the above heuristic according to the nature of the datasets (cyclic and acyclic graphs). The datasets are presented in Table 5.1 and Table 5.2. The results are compared to the ILP solutions already reported in [9]. The ILP model is an exact solution that models the state dynamics of the server during stages of VNFs migrations. The decision variables are binary variables indicating the stage where the VNF is migrated or interrupted. The ILP takes into account the hot and cold migration and the capacity constraint while minimizing the interruption and migration duration.

Table 5.1: Datasets of acyclic graphs

Instances	Servers	VNFs	Slices
DC-acy1	10	25	6
DC-acy2	20	35	11
DC-acy3	40	60	12
DC-acy4	50	120	24
DC-acy5	80	150	35

Table 5.2: Datasets of cyclic graphs

Instances	Servers	VNFs	Slices
DC-cy1	10	30	8
DC-cy2	20	45	12
DC-cy3	40	70	15
DC-cy4	50	120	25
DC-cy5	80	146	32

5.4.2.1 Acyclic graph

In the case we have an acyclic graph the problem is much less complex compared to the cyclic one and we are not reporting all results. In general, we have observed that the algorithm is fast while the ILP model may spend more than 36 seconds for some large instances. With respect to the other two criteria, the PTS solutions remain pretty close to those obtained with ILP, especially for the small and medium instances.

5.4.2.2 Cyclic graph

Fig. 5.5 shows the evolution of the computation time according to different instances. It can be seen that the heuristic solves the problem rapidly in a few milliseconds for different sizes of graphs. Using the ILP, the instance DC-cy5 takes more than one hour while solving the problem with the heuristic is achieved in less than one second. This may be important, especially when dealing with urgent demands of reconfiguration.

The total interruption duration is presented in Fig. 5.6. The interruption duration of the first three instances (DC-cy1, DC-cy2, DC-cy3) for the heuristic is near to that of the ILP. Fig. 5.7 shows the total migration duration of the whole process for the heuristic and the ILP model. We notice that the solutions for the heuristic remain comparable with the optimal solutions of the ILP for small and

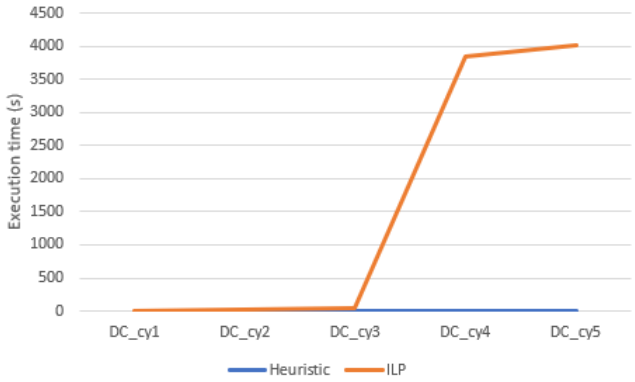


Figure 5.5: Computation time for cyclic graph

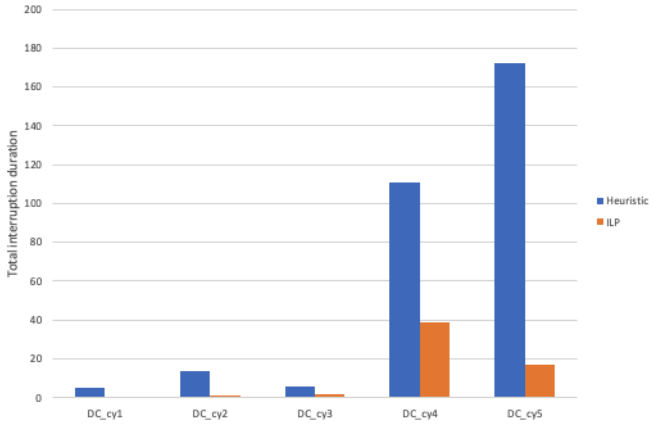


Figure 5.6: The total interruption duration for the cyclic graph

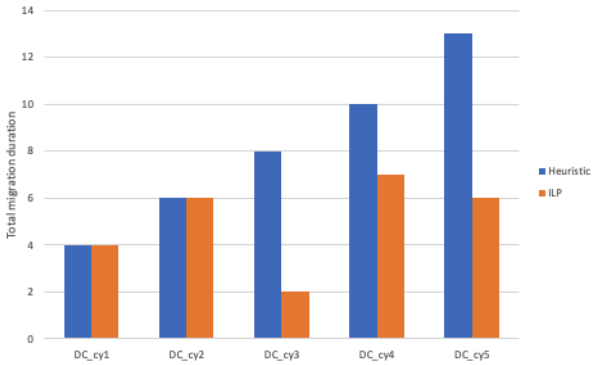


Figure 5.7: The total migration duration for the cyclic graph

medium instances.

5.5 Conclusion

In this work, an efficient heuristic was proposed for the problem of VNF reconfiguration in the context of 5G networks. The proposed heuristic finds the reconfiguration plan, consisting of a series of migrations that will relocate the VNFs from their current servers to those computed beforehand while minimizing the migration and interruption duration. The proposed heuristic was compared to the ILP model presented in [9]. The results show that the heuristic yields good solutions that are close to the optimal solution for small and medium instances. Furthermore, our heuristic finds the reconfiguration plan in milliseconds for large instances, which is not the case when applying the ILP model. Still, we may notice that there is room for improvement, especially when dealing with the interruption time issue for large instances in cyclic graphs.

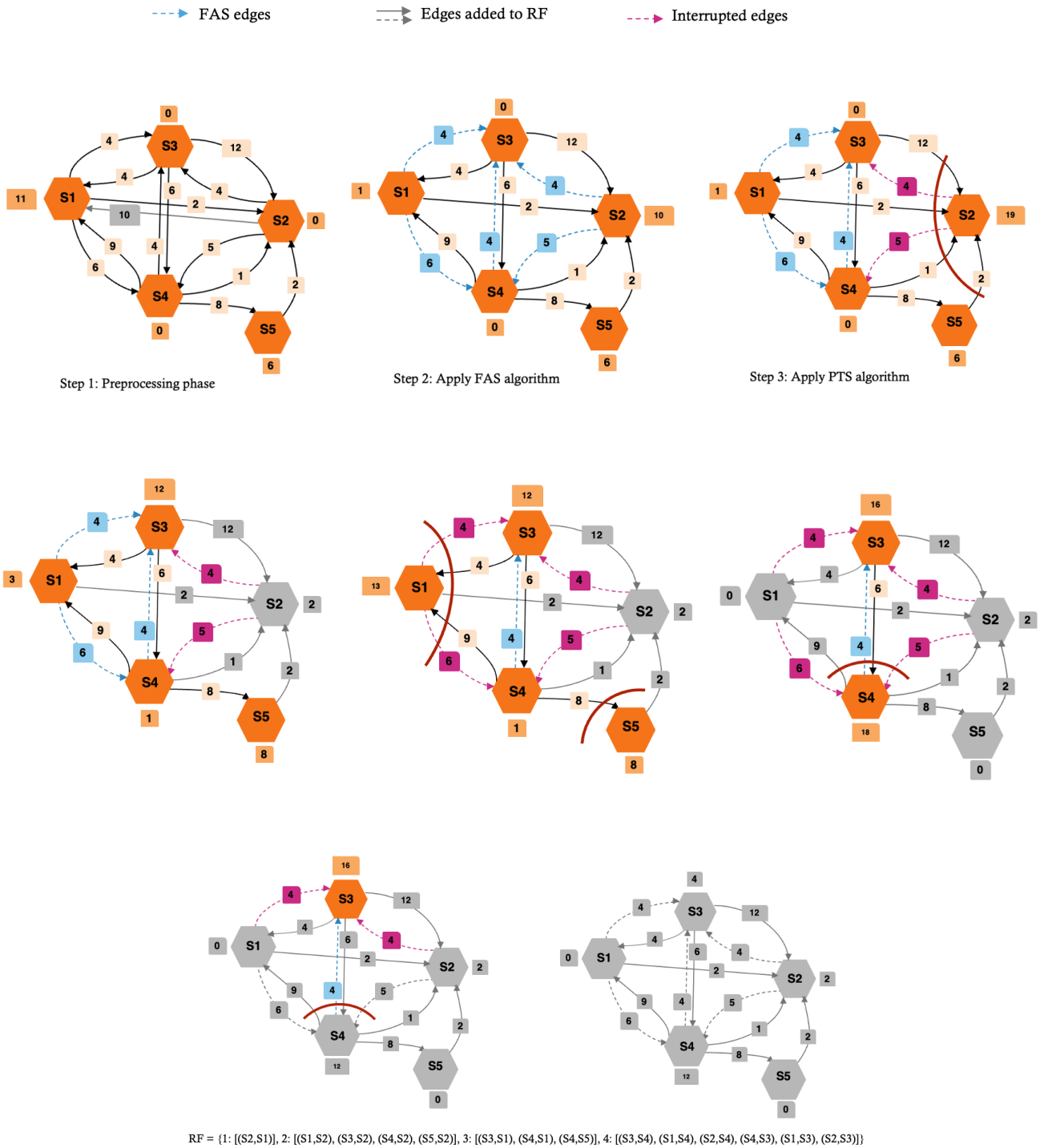


Figure 5.8: Example of the entire process

Chapter 6

Column generation model

Contenu

6.1 Multiple Multidimensional Knapsack Problem 88
6.2 Solution methods for fixed final target 91
6.2.1 First compact formulation 91
6.2.2 Non-compact formulation 92
6.2.3 Second compact formulation 97
6.3 Solution methods for free final target 97
6.3.1 Column generation 98
6.3.2 Second ILP formulation 98
6.4 Numerical Results 99
6.4.1 Fixed final target 99
6.4.2 Free final target 103
6.5 Conclusion 107

This chapter looks at the VNF migration problem. We assume that the initial and final states that define the assignment of the VNFs to their source and destination servers are known. The goal is to define a sequence of steps in the migration from the initial to the final state. At each step, VNFs can only be moved from their source server and/or to their destination server. The objective is to minimize the total time when VNFs are not assigned to a server and therefore must be suspended. Another version of this problem is where the destination state is not fixed, and so defining the final assignment of the VNFs to servers becomes an additional part of the problem. All this is encapsulated in what we call the Multiple Multidimensional Knapsacks Reconfiguration (MMKR) problem, which is the focus of this contribution; the multiple knapsacks correspond to servers, and the multiple dimensions are different capacity-related characteristics of the VNFs and the server.

The contribution of this chapter consists in proposing a new method of solving the MMKR problem, based on a non-compact Integer Linear Programming (ILP) problem formulation and Column Generation, together with an alternative new compact ILP formulation. In addition, we propose an ILP formulation that enables reconfiguring the VNFs in the case where the destination is unknown. The VNFs migrate from the current state to another optimal state that the model ILP computes. We illustrate the resulting effectiveness of solving the problem with a series of numerical experiments, comparing the results to the results of applying the compact ILP problem model reported in Chapter4.

6.1 Multiple Multidimensional Knapsack Problem

The KNAPSACK problem in its basic form has a prominent place in the study of Integer Programming (IP) models with binary variables. The problem consists in packing a given set of items with given values and sizes (such as weights or volumes) into a container with a given maximum capacity C. With the total size of the items potentially exceeding the capacity of the container, usually, it is not possible to pack all of them. The problem is therefore to select a subset of items with the greatest possible total value to place in the container. The ILP formulation of the problem uses one binary decision variable per item to indicate the items that are selected.

A variant of the standard KNAPSACK problem is the MULTIPLE KNAPSACKS problem. Instead of a single knapsack, here there are multiple knapsacks, where knapsack k ($k = 1 \dots K$, K being the number of knapsacks) has capacity C_k. Not only must be selected the items to pack but it must also

be decided which knapsack each item should be placed in.

Another often-studied version of the standard KNAPSACK problem is the MULTIDIMENSIONAL KNAPSACK problem. In this variant of the problem, the set of items selected for packing into the knapsack should satisfy not one but several capacity constraints corresponding to individual capacity dimensions. And putting together multiple dimensions and multiple knapsacks gives rise to the MULTIPLE MULTIDIMENSIONAL KNAPSACKS problem, a problem that has so far been very little addressed in the literature, but which appears to be NP-hard and computationally challenging.

Finally, a reconfiguration aspect can be introduced into the KNAPSACK problem. The idea behind reconfiguration is to start from a given initial assignment of items to knapsacks and to sequentially move items between knapsacks so as to reach a given final assignment. Including reconfiguration brings out the scheduling aspect of the problem which, in conjunction with multiple multidimensional knapsacks, results in the MMKR problem that is our concern here. It is a novel version of the problem, and it turns out to be considerably more difficult.

Arguably, versions of the problem examined in this work have already appeared in other contexts linked to computer networking. Sirdey et al. [29] looked at a process move problem that arises in relation to the operability of a certain class of high-availability real-time distributed systems. Given an initial and a final system state defining which processes are assigned to which processors in a distributed system, the goal is to find a minimally disruptive sequence of operations (non-impacting process migrations or temporary process interruptions) at the end of which the system is in the final state; at no step can the capacity of any processor be exceeded. Thinking of each processor as a knapsack and of each process as an item, this problem has a lot in common with the MMKR problem. The main difference lies in the technical context: in our 5G VNF reconfiguration problem migrations are realized according to rules (known as hot and cold migration of VNFs) that impose specific constraints upon feasible migration sequences.

Solano and Pióro [78] studied the problem of lightpath reconfiguration in optical transport networks. Lightpath reconfiguration is a networking task that is performed in order to improve resource utilization and limit network congestion. The problem consists in finding an optimal transition from an initial set of currently operational lightpaths to a new set of lightpaths guaranteeing better network performance. The problem becomes nontrivial when establishing the new set of lightpaths requires the release of resources held by the currently operational lightpaths while ensuring continuity of traffic

flows (an operational lightpath cannot be removed before the corresponding new lightpath is set up). The reconfiguration scheduling seeks to make the maximum number of simultaneously disrupted connections at any point during the process as small as possible [78]. Although the application context is different from that of the MMKR problem, the formulations of the two problems have a lot in common. The existing and the new lightpath connections can be seen as items migrating over network links which, being bundles of optical wavelengths, may be considered as knapsacks, insofar as they provide the resources for the connections.

There has also been a lot of work done on problems that overlap with the MMKR problem. [79] proposes a resource allocation and management mechanism for 5G networks using Multi-access Edge Computing (MEC) technology. A simple mathematical model of the MEC resource (re)allocation mechanism is proposed to meet the requirements of the user.

[80] considers a control problem in the reconfiguration of photovoltaic arrays that involves changing the connections in the solar panel equipped with the dynamic switching matrix. The proposed formulation of the problem is based on the well-known subset sum problem, which is a special case of the Knapsack problem. The solution method uses a dynamic programming algorithm that is capable of computing an optimum reconfiguration. The characteristic feature of this problem is that the final configuration of the array is not given.

[81] considers centralized spectrum allocation in cognitive radio networks as a Multiple Multidimensional Knapsacks problem. The formulation of the problem models the behavior of the Centralized Coordinator Node (CCN) that shares spectrum availability information with a set of cognitive users. Each primary user acts as a two-dimensional knapsack, with bandwidth and temperature interference as the limiting resources. CCN has to assign the cognitive users, considered as items, to primary users in such a way that the overall traffic throughput is maximized while the resource constraints are satisfied. The proposed formulation is similar to our problem formulation, but it is limited to the computation of a static assignment of items to knapsacks and does not include the scheduling aspect.

6.2 Solution methods for fixed final target

In this section, we discuss the exact methods of solving the MMKR problem for fixed final target, where the VNF placement in the target state is known beforehand. These methods are essentially based on Integer Linear Programming. Table 6.1 lists the notation used.

6.2.1 First compact formulation

In this section, we present a compact ILP model inspired from the model reported in Chapter 4. We use the following variables. For each $i \in I$, x_i is a continuous positive variable denoting the interruption time of item i. For all $i \in I$, $t \in T$, y_i^t is a binary variable that equals 1 if, and only if, item i is interrupted at its source at time t, and z_i^t is a binary variable that equals 1 if, and only if, item i is placed at its destination at time t. For all $i \in I$, $t \in T$, c_k^t is a continuous positive variable denoting the residual capacity of knapsack k at time t. The MMKR problem can then be written as:

$$\min \sum_{i \in \mathcal{I}} w(i) x_i \tag{6.1a}$$

$$\sum_{i \in \mathcal{O}(k)} w(i) + c_k^1 = C(k) \qquad \forall k \tag{6.1b}$$

$$\sum_{i \in \mathcal{O}(k)} w(i) y_i^t - \sum_{i \in \mathcal{D}(k)} w(i) z_i^t + c_k^t = c_k^{t+1} \qquad \forall k,\ \forall t < T \tag{6.1c}$$

$$\sum_{t \in \mathcal{T}} y_i^t = 1 \qquad \forall i \tag{6.1d}$$

$$\sum_{t \in \mathcal{T}} z_i^t = 1 \qquad \forall i \tag{6.1e}$$

$$\sum_{i \in \mathcal{O}(k)} y_i^t + \sum_{i \in \mathcal{D}(k)} z_i^t \leq 1 \qquad \forall k,\ \forall t \tag{6.1f}$$

$$\sum_{t \in \mathcal{T}} t z_i^t - \sum_{t \in \mathcal{T}} t y_i^t + 1 \leq x_i \qquad \forall i \tag{6.1g}$$

$$x_i \geq 0 \qquad \forall i \tag{6.1h}$$

$$y_i^t, z_i^t \in \{0, 1\} \qquad \forall i,\ \forall t \tag{6.1i}$$

$$c_k^t \geq 0 \qquad \forall k,\ \forall t. \tag{6.1j}$$

The objective is to minimize the weighted interruption time over all items. Constraints (6.1g) force the interruption time of the item to be greater than the difference between the time of placing the

Table 6.1: Notation

K	the number of knapsacks
$\mathcal{K}$	the set of knapsacks, $\mathcal{K} = \{1, 2, \ldots, K\}$
k	a specific knapsack
$C(k)$	the resource capacity of knapsack k
I	the number of items
$\mathcal{I}$	the set of items, $\mathcal{I} = \{1, 2, \ldots, I\}$
i	a specific item
$o(i)$	the source knapsack of item i
$d(i)$	the destination knapsack of item i
$w(i)$	the resource requirement of item i
$\mathcal{O}(k)$	the items with knapsack k as source, $\mathcal{O}(k) \subseteq \mathcal{I}$
$\mathcal{D}(k)$	the items with knapsack k as destination, $\mathcal{D}(k) \subseteq \mathcal{I}$
T	the number of time periods
$\mathcal{T}$	the set of time periods, $\mathcal{T} = \{1, 2, \ldots, T\}$
t	a specific time period

item at the destination and the time of interrupting at the source (increased by 1 because the VNF is not available until the next time period after transfer); we note that the nature of the objective forces this constraint to equality.

Unfortunately, formulation (6.1) does not scale well and our goal in this work is to propose alternative solution methods. The following is a method based on Column Generation.

6.2.2 Non-compact formulation

To use the Column Generation method, we need an alternative non-compact problem formulation. The main challenge is deciding on the format of the columns. We are seeking to represent (over the relevant time periods) the movement of an item from the source knapsack to the destination knapsack as a path in a graph with (knapsack, time) tuples as nodes. For each knapsack and each time period, there will be a specific node in the graph, and at each time period the item is to be found in at least one knapsack, and in at most two (there are two knapsacks when the migration occurs). A path in this graph, therefore, represents the placement of the item over all time periods. The arcs (and consequently the paths) allowed in the graph should reflect all possible item migrations. We have T

time periods and $K+1$ knapsacks replicated at each period, the knapsack $(K+1)$ being a virtual knapsack of infinite capacity, containing all the interrupted items. If an arc enters a node (k,t) the corresponding item consumes resources of knapsack k at time period t. Since the number of possible paths is large, they cannot all be included in the problem formulation, and the Column Generation method deals neatly with this situation. We start with a basic feasible set of paths and then search for a new path that improves the objective function.

Each column takes the form $a = (a_{1,1}, \ldots, a_{K+1,1},\ a_{1,2}, \ldots, a_{K+1,2}, \ldots, a_{1,T}, a_{2,T}, \ldots, a_{K+1,T})$, where $a_{k,t}$ is 1 if item i is placed at knapsack k at time t, and 0 otherwise.

Each of these columns represents a specific path p that some item i can use to migrate from its source to its destination knapsack. Since at the outset there will be only a small set of paths guaranteeing an initial feasible solution, we need to search for new paths (columns) in the above format to be included in the path set P_i for item i, with a view to obtaining a better objective function value. When searching for such a column all its elements $a_{k,t}$ will be set as binary variables. In addition, each column in the Master Problem has a corresponding binary path variable y_{ip} that indicates whether path p for item i is used (y_{ip}=1) or not used (y_{ip}=0).

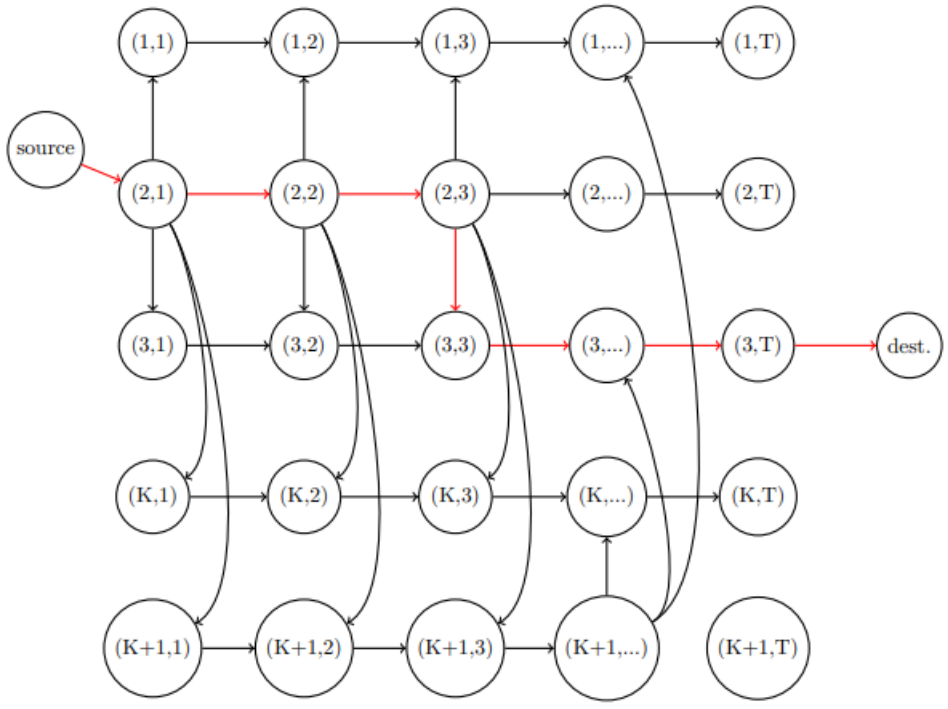


Figure 6.1: Graph representation

The graph in Figure 6.1 shows a migration from source knapsack 2 to destination knapsack 3 at a time period of 3. At a time period, 3 resources are thus consumed in both knapsack 2 and knapsack 3. For the sake of clarity, only a subset of arcs is shown just to give an idea of the allowed arcs and paths. We can move from the source knapsack to other knapsacks during each time period except the last one. If this move is to the destination knapsack, then the item will stay there until the end. If the move is to the interruption knapsack, then we are forced to migrate back to the destination knapsack before the last time period, with vertical moves coming out from knapsack $(K+1)$.

6.2.2.1 Master Problem

Now that we have a better theoretical grasp of the context of the problem, we are able to put forward a corresponding non-compact Linear Programming (LP) formulation as the first step in our column generation algorithm. For sake of simplicity, the one-dimensional case is assumed.

$$\min \sum_{t\in\mathcal{T}} \sum_{i:(K+1,t)\in p, p\in\mathcal{P}_i} w(i)y_{ip} \tag{6.2a}$$

$$\sum_{i:(k,t)\in p, p\in\mathcal{P}_i} w(i)y_{ip} \leq C(k) \qquad \forall k,\ \forall t \qquad (\lambda_{kt}) \tag{6.2b}$$

$$\sum_{p\in\mathcal{P}_i} y_{ip} = 1 \qquad \forall i \qquad (\gamma_i) \tag{6.2c}$$

$$y_{ip} \geq 0. \tag{6.2d}$$

The objective function aims at minimizing the overall path weights going through the interruption knapsack summed over all time periods, which is equivalent to minimizing the total interruption cost. For a specific time period t the second sum iterates over all items that have $(K+1,t)$ in a path p belonging to their path set $\mathcal{P}_i$.

The first constraint ensures that knapsack capacities are always satisfied during all time periods. We sum over all weighted paths going through each node *(k,t)* and require that the capacity of that node is never exceeded.

the second constraint helps to pose the relaxed (continuous) version of the problem because in the binary version only one y_{ip} for an item i is equal to 1 and all other variables should equal 0. For each such constraint, the dual variable is indicated beside them (λ_{kt}, γ_i, respectively).

6.2.2.2 Pricing Problem

The second step in the algorithm requires defining the Pricing Problem. As mentioned above, the Pricing Problem will be an Integer Linear Programming using the dual variables found in the first step as parameters and column elements a_{kt} as variables.

$$\min\ w(i) \sum_{k\in\mathcal{K}} \sum_{t\in\mathcal{T}} \lambda_{k,t} a_{k,t} - \gamma_i \equiv \tag{6.3a}$$

$$\min \sum_{k\in\mathcal{K}} \sum_{t\in\mathcal{T}} \lambda_{k,t} a_{k,t} \equiv \tag{6.3b}$$

$$\min \sum_{(k,t)\in p\in\mathcal{P}_i} \lambda_{k,t} \tag{6.3c}$$

$$\sum_{k\in\mathcal{K}} a_{k,t} \geq 1 \qquad \forall t \tag{6.3d}$$

$$a_{k,t} \in \{0,1\} \qquad \forall k,\ \forall t. \tag{6.3e}$$

The above model is complemented with the following constraint if no interruptions are allowed

$$\sum_{t\in\mathcal{T}} \sum_{k\in\{1,\ldots,K\}} a_{k,t} \leq T+1, \tag{6.4}$$

and with the following one if interruptions are allowed

$$\sum_{t\in\mathcal{T}} \sum_{k\in\{1,\ldots,K+1\}} a_{k,t} \leq T+2. \tag{6.5}$$

Indeed, the above constraints express the length of a path which is connected to the number of interruptions. the objective function above is a graph shortest path problem, where the arc weights are denoted by dual variables $\lambda_{k,t}$. We are searching for the shortest path from source to destination knapsack in a directed graph where each incoming arc at node (k,t) has a weight $\lambda_{k,t}$.

The condition (6.3d) indicates that an item should be placed in at least one knapsack at each time period.

The conditions (6.4) and (6.5) restrict the movements of items between knapsacks. Only one knapsack change, denoted by a vertical move in the graph, is allowed if there are no interruptions (i.e., knapsack $(K+1)$ is not used). But, if we are searching for a path with interruption (i.e., knapsack $(K+1)$ is used), then we allow at most two vertical moves.

It is not by chance that the Pricing Problem is the shortest path problem. Usually, this problem is an ILP with a special combinatorial structure: knapsack problem, shortest path problem, maximum flow problem, or other problems of this nature. The success of applying column generation often relies on efficient algorithms to solve the pricing problem. For that reason, we usually search for a combinatorial algorithm to efficiently solve the problem rather than posing the ILP version of the problem to the solver. The algorithm allows to directly exploit all the details specific to the problem, while the solver follows a more general approach to find the solution. Based on this fact, the shortest path algorithm comes naturally as the combinatorial algorithm for minimizing the objective function.

6.2.2.3 Achieving an integer solution

As the final step in the column generation algorithm, we need to obtain an integer solution to the problem. In this section, we present the different methods:

- Final ILP: One good way of not shifting the results too much in relation to the continuous results is to solve the final Master Problem as an Integer Linear Programming problem. In this final Master Problem, we use the final set of paths found by iteratively solving the continuous version of Column Generation until optimality. The path variables are then forced to binary. This method is referred as Solving the Final Master Problem (*Final ILP*).

- Post optimization: The main idea of this method is to start with an initial solution obtained by rounding and to add new paths until a possible solution is reached. This initial solution may be infeasible after rounding the variables. We, therefore, identify the knapsack in which the capacity constraint has been violated and the items that it contains. We then modify the paths of some of the items in the knapsack by adding interruptions in them to make the solution feasible. This is noted with *PostOpt*.

- A mixed method: Another method involves solving the Master Problem one more time but now with the addition of the paths established using the heuristic method. A comparison of the objective functions of the different methods shows that solving the Master Problem once again in this way gives us the lowest cost. The ILP version of the final Master Problem (after the addition of paths identified by the post-optimization method) is consequently the best version,

maintaining feasibility and having the lowest interruption cost, that is to say, the closest to the continuous interruption cost. This method is noted as *PostOpt + Final ILP*.

6.2.3 Second compact formulation

Below we introduce a new compact ILP model of the MMKR problem, the origin of this model is Artur Tomaszewski [11]. We define the following variables. For all $i \in \mathcal{I}$, $t \in \mathcal{T}$, x_i^t is a binary variable that equals 1 if, and only if, item i is at its source at time t, y_i^t is a binary variable that equals 1 if, and only if, item i is at its destination at time t, and z_i^t is a binary variable that equals 1 if, and only if, item i is interrupted at time t. Then, using the observable monotonicity of the values of variables x and y with time periods, the MMKR problem can be formulated as:

$$\min \sum_{i\in\mathcal{I}}\sum_{t\in\mathcal{T}} w(i)z_i^t \tag{6.6a}$$

$$x_i^t \geq x_i^{t+1} \qquad \forall i,\ \forall t < T \tag{6.6b}$$

$$y_i^t \geq y_i^{t-1} \qquad \forall i,\ \forall t > 1 \tag{6.6c}$$

$$x_i^t + y_i^t + z_i^t \geq 1 \qquad \forall i,\ \forall t \tag{6.6d}$$

$$\sum_{t\in\mathcal{T}} (x_i^t + y_i^t + z_i^t) \geq T + 1 \qquad \forall i \tag{6.6e}$$

$$\sum_{i\in\mathcal{O}(k)} w(i)x_i^t + \sum_{i\in\mathcal{D}(k)} w(i)y_i^t \leq C(k) \qquad \forall k,\ \forall t \tag{6.6f}$$

$$x_i^t, y_i^t, z_i^t \in \{0,1\} \qquad \forall i,\ \forall t. \tag{6.6g}$$

Once again, the objective is to minimize the weighted interruption time over all items. While constraints (6.7f) require that each item is either in a knapsack or is interrupted, constraints (6.7g) result from the mentioned specifics of VNF migration that the VNF is not available until the next time period after the transfer, and require that after the item has been moved to the destination knapsack and the item is not still in the source knapsack it must be regarded as interrupted.

6.3 Solution methods for free final target

In this section, we discuss the exact methods of solving the MMKR problem for the free final target, where The VNFs placement in the target state is unknown. The objective is to generate a re-

configuration plan that ensures the transition between the states and enables optimal VNF placements in the target state.

6.3.1 Column generation

For the Column Generation formulation, the model will be the same just keeping in mind that now the path that will be added as a new column will be the shortest path possible knowing just the origin for each item and every knapsack could be as a destination.

6.3.2 Second ILP formulation

We define the following variables. For all $i \in \mathcal{I}$, $t \in \mathcal{T}$, x_i^t is a binary variable that equals 1 if, and only if, item i is at its source at time t, $y_{i,k}^t$ is a binary variable that equals 1 if, and only if, item i is a at knapsack k at time t, u_k^i is a binary variable that equals 1 if, and only if, knapsack k is a destination for item i, and z_i^t is a binary variable that equals 1 if, and only if, item i is interrupted at time t. Then, using the observable monotonicity of the values of variables x and y with time periods, the MMKR problem can be formulated as:

$$\min \sum_{i \in \mathcal{I}} \sum_{t \in \mathcal{T}} w(i) z_i^t \tag{6.7a}$$

$$\sum_{\forall k} u_i^k = 1 \qquad \forall i \tag{6.7b}$$

$$y_{i,k}^t \leq u_i^k \qquad \forall i \tag{6.7c}$$

$$x_i^t \geq x_i^{t+1} \qquad \forall i,\ \forall t < T \tag{6.7d}$$

$$\sum_{\forall k} y_{i,k}^t \geq \sum_{\forall k} y_{i,k}^{t-1} \qquad \forall i,\ \forall t > 1 \tag{6.7e}$$

$$x_i^t + \sum_{\forall k} y_{i,k}^t + z_i^t \geq 1 \qquad \forall i,\ \forall t \tag{6.7f}$$

$$\sum_{t \in \mathcal{T}} (x_i^t + \sum_{\forall k} y_{i,k}^t + z_i^t) \geq T + 1 \qquad \forall i \tag{6.7g}$$

$$\sum_{i \in \mathcal{O}(k)} w(i) x_i^t + \sum_{i \in \mathcal{I}} w(i) y_{i,k}^t \leq C(k) \qquad \forall k,\ \forall t \tag{6.7h}$$

$$x_i^t, y_{i,k}^t, z_i^t \in \{0, 1\} \qquad \forall i,\ \forall t,\ \forall k. \tag{6.7i}$$

6.4 Numerical Results

This section is devoted to numerical results. We ran our methods on realistic problem instances, using randomly generated topologies to build acyclic and cyclic graphs. The graphs, of different sizes (small and medium), were generated randomly using NetworkX, (a Python lib), based on [71]. The nodes of the graph represent the servers, and the links represent the VNF migration. The node and link capacities were generated randomly, $(1 - 50)$ for CPU capacity and $(10 - 90)$ for RAM capacity. The type number of the VNFs were randomly generated in the range $(20 - 220)$ and the number of servers in the range $(10 - 100)$. The datasets are presented in Table 6.2. Another element included in the tables is the number of periods (Periods) allowed for the migration. This aspect may be important in some situations where the migrations have to be realized in the shortest possible time. In theory, the maximum number of periods is bounded by the number of migrations (items), but in practice, several migrations can be processed in parallel. An assessment of the Column Generation method together with its several post-optimization methods for both cyclic and acyclic graphs can be seen in Table III and Table IV, respectively. It will be remarked that the mixed method (that is PostOpt + Final ILP) performed significantly better. This method is used below for the purposes of comparison with the ILP formulations.

Table 6.2: Datasets of cyclic and acyclic graphs

	Cyclic Graphs			Acyclic Graphs		
Instance	Server	Item	Periods	Server	Item	Periods
1	10	30	$I/4$	10	25	I
2	20	45	$I/5$-1	20	35	$I/4$
3	40	70	$I/8$	40	60	$I/9$
4	50	90	$I/10$	50	80	$I/10$
5	80	146	$I/18$	80	150	$5I/25$
6	100	220	$I/32$-1	100	200	$I/30$

6.4.1 Fixed final target

In this section, we present the evaluation results for the case where the final target is known for cyclic and acyclic graphs.

6.4.1.1 Cyclic graph

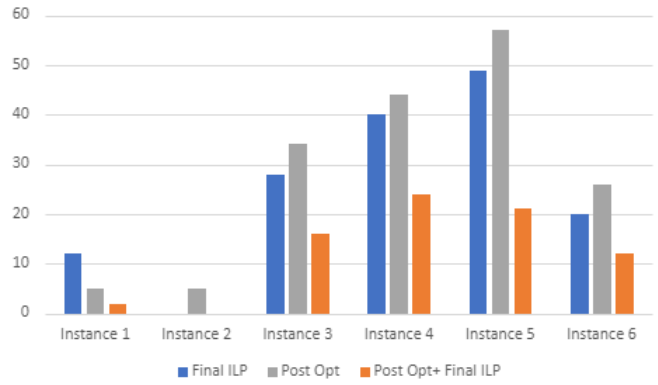


Figure 6.2: Interruption cost of cyclic graphs (Column Generation)

In the following, we report comparative numerical results for the three methods, namely column generation completed with mixed post optimisation method (Post Opt+ Final ILP), first compact ILP, and second compact ILP. The notation TL stands for the out of time limit, the limit being set to 3600 seconds.

Table 6.3: Interruption cost of cyclic graphs for all models

Inst	Server	Item	Periods	Final ILP	Post-Opt	Post-Opt + Final ILP	First ILP Formulation	Second ILP Formulation
1	10	30	$I/4$	12	5	2	0	0
2	20	45	$I/5$-1	0	5	0	0	0
3	40	70	$I/8$	28	34	16	TL	2
4	50	90	$I/10$	40	44	24	TL	0
5	80	146	$I/18$	49	57	21	TL	0
6	100	220	$I/32$-1	20	26	12	TL	0

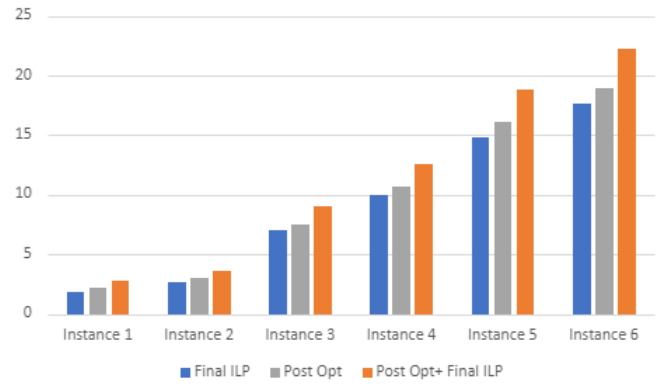


Figure 6.3: Execution time of cyclic graphs in seconds (Column Generation)

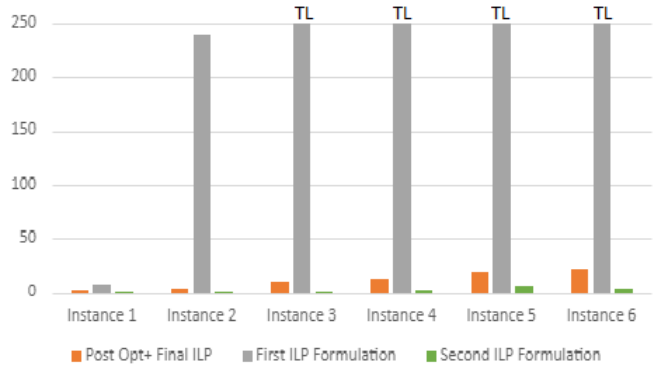


Figure 6.4: Execution time of cyclic graphs in seconds

Fig. 6.2 presents the interruption cost of the cyclic graphs for the column generation method. The results show that the mixed method has the lowest interruption cost comparing to the Final ILP and post optimization methods, but it takes more time to find the optimal solution (see Fig. 6.3). Table 6.3 summarizes the experimental results of the two compact ILP. The second ILP formulation gives better results comparing to the first ILP model and column generation method in terms of interruption cost and execution time as shows the Fig. 6.4.

6.4.1.2 Acyclic graph

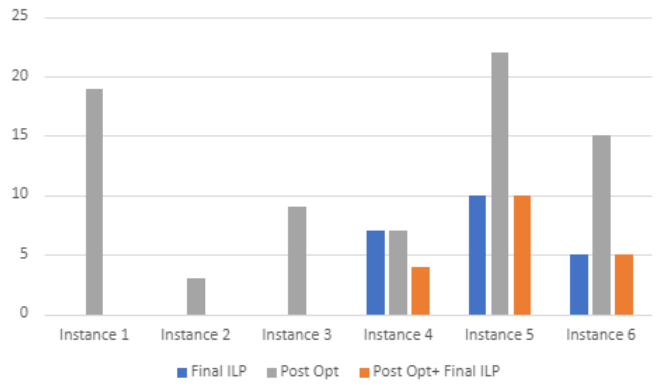


Figure 6.5: Interruption cost of acyclic graphs (Column Generation)

Table 6.4: Interruption cost of acyclic graphs for all models

Inst	Server	Item	Periods	Final ILP	Post-Opt	Post-Opt + Final ILP	First ILP Formulation	Second ILP Formulation
1	10	25	I	0	19	0	0	0
2	20	35	$I/4$	0	3	0	0	0
3	40	60	$I/9$	0	9	0	0	0
4	50	80	$I/10$	7	7	4	0	0
5	80	150	$I/25$	10	22	10	TL	0
6	100	200	$I/30$	5	15	5	TL	0

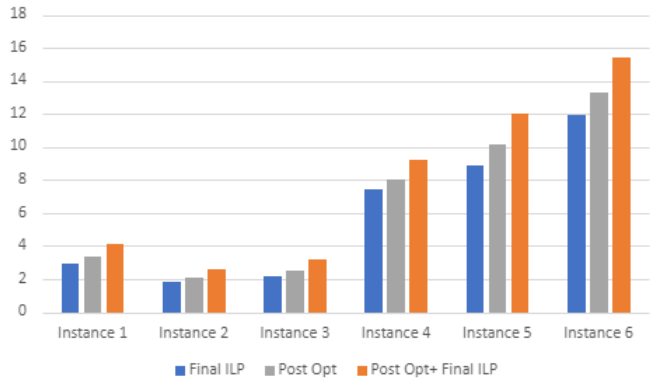


Figure 6.6: Execution time of acyclic graphs in seconds (Column Generation)

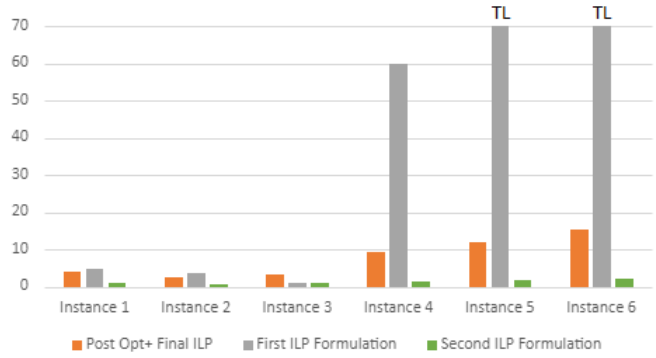


Figure 6.7: Execution time of acyclic graphs in seconds

From the results above it may be concluded that the Column Generation method gives better results than the first compact formulation, while keeping the interruption cost under control. In the case of acyclic graphs, the cost is zero in most cases. However, it will also be remarked that the second compact formulation achieves the optimal solution in a very short time. Nevertheless, the column generation method has the advantage of offering more possibilities as regards incorporating the sort of difficult new constraints that are encountered in real-world conditions. These constraints may concern the overall duration of the migration process which needs to be minimized, or specific interruption options that are to be included in the main problem.

6.4.2 Free final target

In this section, we present the evaluation results for the case where the final target is unknown for cyclic and acyclic graphs.

6.4.2.1 Cyclic Graphs

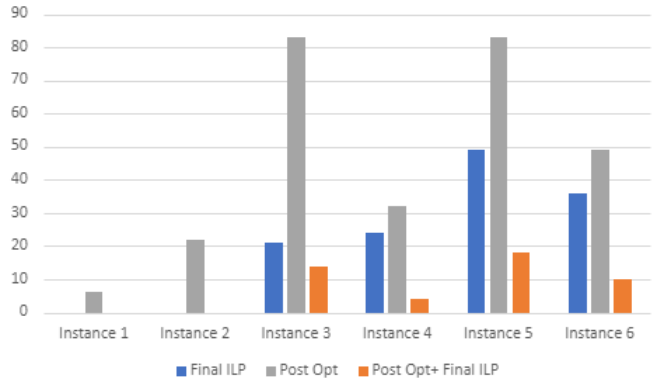


Figure 6.8: Interruption cost of cyclic graphs (Column Generation)

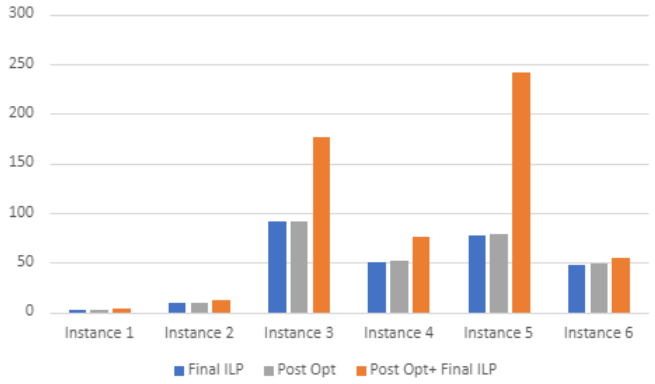


Figure 6.9: Execution time of cyclic graphs in seconds (Column Generation)

As can be seen from the graphs, solving the Master Problem one more time, but now with the addition of the paths established using the heuristic method, requires more time, because now we have more possibilities to choose the path for each item.

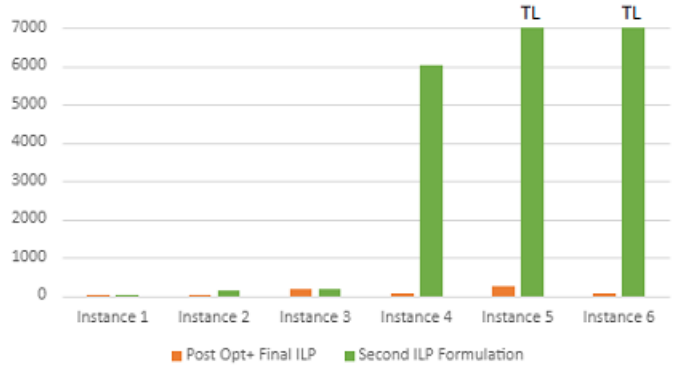


Figure 6.10: Execution time of cyclic graphs in seconds

Table 6.5: Interruption cost of cyclic graphs for all models

Inst	Server	Item	Periods	Final ILP	Post-Opt	Post-Opt + Final ILP	Second ILP Formulation
1	10	30	$I/4$	0	6	0	0
2	20	45	$I/5$-1	0	22	0	0
3	40	70	$I/8$	21	83	14	0
4	50	90	$I/10$	24	32	4	0
5	80	146	$I/18$	49	83	18	TL
6	100	220	$I/32$-1	36	49	10	TL

6.4.2.2 Acyclic Graphs

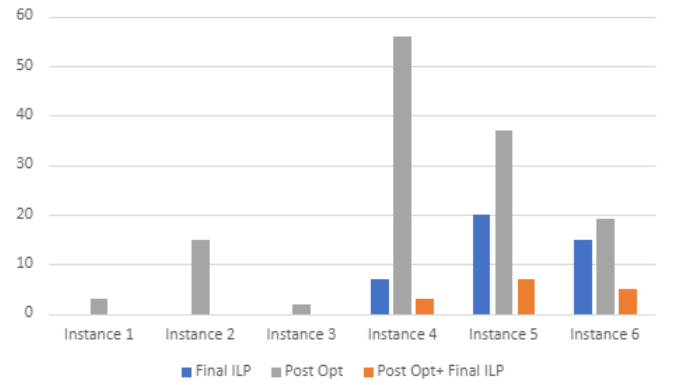


Figure 6.11: Interruption cost of acyclic graphs (Column Generation)

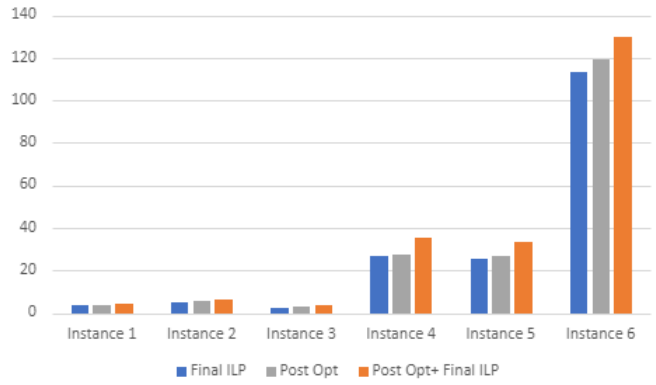


Figure 6.12: Execution time of acyclic graphs in seconds (Column Generation)

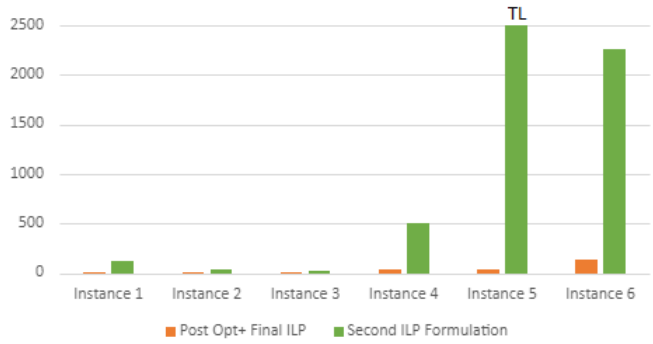


Figure 6.13: Execution time of acyclic graphs in seconds

Table 6.6: Interruption cost of acyclic graphs for all models

Inst	Server	Item	Periods	Final ILP	Post-Opt	Post-Opt + Final ILP	Second ILP Formulation
1	10	25	I	0	3	0	0
2	20	35	$I/4$	0	15	0	0
3	40	60	$I/9$	0	2	0	0
4	50	80	$I/10$	7	56	3	0
5	80	150	$I/25$	20	37	7	TL
6	100	200	$I/30$	15	19	5	0

As can be seen from the tables, compared to the Second ILP Formulation, the Column Generation Method gives faster results, while keeping the interruption cost under control. The second ILP For-

mulation, in fact, provides the interruption with a cost equal to zero in all the cases, because now the model is more accessible in choosing the destination for each item. The problem in this formulation is that we add a lot of variables because we have the variable $y^t_{i,k}$, but also the fact that our variables are integer values, makes the problem complex and requires more time.

6.5 Conclusion

In this chapter, we have presented some new results on an optimization problem relating to resource allocation in 5G networks. We model the problem as Multiple Multidimensional Knapsacks Reconfiguration problems and formulate it through Integer Linear Programming compact and non-compact models. In addition, we propose an ILP model for the case where the destination is unknown.

One of the main issues here is that solution choices have a high combinatorial potential, which may make the optimization problem computationally intractable for large problem instances. Indeed, from the experimental results, we see that the first compact formulation becomes intractable when the number of items and the number of servers increases. We propose a Column Generation solution approach.

The proposed non-compact ILP problem model and the corresponding solution method are easily adaptable to the case where the destination state is not fixed and needs to be determined. The method does not, however, yield systematically integer solutions, and it consequently needs to be supplemented by a post-optimization step, which does not invariably lead to an optimal solution either. Nevertheless, the method exhibits shorter computation times than the first compact formulation, although computation times remain significantly higher than those exhibited by the second compact formulation.

To conclude, we use the column generation algorithm in order to not generate and use the whole set of paths because of the enormous number of them. It brings the advantage of solving larger instances and taking less computational time to find the solution compared to a pure ILP approach.

Chapter 7

Conclusion

Contenu

7.1 Thesis contributions: a summary 110

7.2 Future work and perspectives 111

7.1 Thesis contributions: a summary

In this thesis, we have defined and studied the problem of VNF reconfiguration in the context of 5G network slicing, for which we have proposed mathematical models and efficient algorithms to solve it. A first contribution is the study on the industrial context from where the problem is extracted. A state of the art is also provided. Next, we proposed, in Chapter 4, an ILP formulation that enables to schedule the VNFs and generates a reconfiguration plan to pass from an initial state where the resources of slices are not optimal to a new state where the optimality is assured while minimizing the service interruption, the migration duration and respecting the resource capacity. The reconfiguration plan is a list of actions (migrations) that will relocate the VNFs from their current servers to those computed beforehand. This problem is shown to be NP-complete in the strong sense [29] [82]. We took into consideration two types of migrations, hot and cold migrations. We evaluate the proposed model according to the service importance taking into consideration the SLA availability metric, and according to the nature of the datasets (whether it is represented by an acyclic or a cyclic graph). The drawback is that the ILP spends a significant amount of time solving large instances. To overcome this problem, we proposed a new heuristic in Chapter 5, in order to improve the computational time and allow dealing with larger instances. The heuristic is based on the feedback arc set problem and an efficient heuristic inspired by the concept of a topological sorting algorithm. The heuristic used for the feedback arc set problem is a greedy heuristic for the minimum set cover problem. The proposed heuristic is the parallel topological sorting that enables the migration of the VNFs in parallel. The proposed heuristic was compared to the ILP model presented in Chapter 4. In contrast to the ILP model, the heuristic finds the reconfiguration plan in milliseconds for large instances. Furthermore, the heuristic yields good quality solutions that are close to the optimal for small and medium instances. Nevertheless, the interruption time for large instances in the case of cyclic graphs remain far from optimality and there is room for improvement.

The third contribution of this PhD thesis was the column generation model. We proposed in Chapter 6 a compact and non-compact ILP model for the column generation model as well as an alternative ILP model. First of all, we were able to get results for all testing instances, cyclic and acyclic, even for those that could not be solved in a reasonable amount of time by the first ILP approach proposed in Chapter 4. In fact, column generation is a methodology designed and used for

large instances, so we do not need to take into consideration the whole set of decision variables and corresponding columns. Secondly, the computational time for all instances is no more than 1 second if we use the time periods shrinkage, compared to the ILP computational time. This advantage is not granted, but it depends on the proper formulation of the Master problem and finding a fast combinatorial algorithm for solving the Pricing problem. Still, this method is not an exact one and we cannot guarantee the optimality. Comparing the interruption cost values of the column generation model to the compact ILP formulation, we observed that the interruption cost for a number of instances is some times larger. To conclude, the column generation model brings the advantage of solving larger instances and taking less computational time to find the solution compared to a pure ILP approach. This may be seen as an heuristic approach.

7.2 Future work and perspectives

In this section, we present directions for future work as a perspective for the work conducted during this PhD thesis. We provide two main directions for future work:

- The first direction consists in improving the heuristic proposed in chapter 5. To recall, the limitation found in the proposed heuristic is that the interruption time for large instances in the case of cyclic graphs is significant. To overcome this issue, one approach would be to consider is what we call a warm start or advanced start. It allows providing the CPLEX optimizer an advanced starting point for optimization. The idea consists of providing CPLEX the solution of the heuristic previously solved, which means we can provide to the CPLEX the reconfiguration plan found using the heuristic as an initial solution, in order to improve this solution by using the ILP model presented in Chapter 4. This approach can be used to deal with the heuristic limitation and improve the interruption time by finding the optimal solution by using the ILP model.

- An interesting perspective for the work performed in this PhD thesis consists in studying the precedence relations between the VNFs in the context of service chaining. Dynamic programming is one of the approaches that we can apply to this problem so that the precedence constraints can be satisfied using this approach. To recall, the dynamic programming [83] [84] follows the methodology of the divide-and-conquer method, by combining the solutions of subproblems to

solve the problem. Dynamic programming aims at reusing the previously found solutions, that's why it is appliable when subproblems share subsubproblems. It solves each subproblem just once and then saves the solution in a table, thereby avoiding the work of recomputing the solution every time it solves each subproblem. When developing a dynamic-programming algorithm, we follow a sequence of four steps: (1) Characterize the structure of an optimal solution. (2) Recursively define the value of an optimal solution. (3) Compute the value of an optimal solution, typically in a bottom-up fashion. (4) Construct an optimal solution from computed information. Usually, the value found in step 3 is enough to solve most problems, but sometimes we may need to construct a combinatorial object that lies behind that optimal value. Actually, we have already studied this approach of a dynamic programming algorithm, based on the migration steps of the items (VNFs). The algorithm searches for the sequence of the optimal steps in a tree structure, enumerating all possible scheduling sequences. We tested some instances and have some preliminary results that show that the algorithm requires a considerable amount of memory and computational time, because of the enormous number of states even for relatively small number of migrations. We have obtained encouraging results only in the case where we allowed exclusively the hot migration.

Bibliography

[1] P. Marsch *et al.*, *5G System Design*. Wiley Online Library, 2018.

[2] J. Ordonez-Lucena *et al.*, "Network slicing for 5g with sdn/nfv: Concepts, architectures, and challenges," *IEEE Communications Magazine*, vol. 55, n°. 5, p. 80–87, 2017.

[3] G. ETSI, "Etsi gs nfv 002 network functions virtualization (nfv), architectural framework," ETSI, Tech. Rep. GS NFV 002, Rapport technique, 2014.

[4] A. Nakao *et al.*, "End-to-end network slicing for 5g mobile networks," *Journal of Information Processing*, vol. 25, p. 153–163, 2017.

[5] GSMA, "E2e network slicing architecture," vol. 48, p. 10, 2021. [En ligne]. Disponible: https://www.gsma.com/newsroom/wp-content/uploads//NG.127-v1.0-1.pdf

[6] 3GPP, "5g; procedures for the 5g system (3gpp ts 23.502 version 15.2.0 release 15), v15.2.0." [En ligne]. Disponible: https://www.etsi.org/deliver/etsi_ts/123500_123599/123502/15.02.00_60/ts_123502v150200p.pdf

[7] ——, "5g; service requirements for enhanced v2x scenarios (3gpp ts 22.186 version 16.2.0 release 16), v16.2.0," 2020. [En ligne]. Disponible: https://www.etsi.org/deliver/etsi_ts/122100_122199/122186/16.02.00_60/ts_122186v160200p.pdf

[8] NGMN, "Ngmn 5g white paper," 2015. [En ligne]. Disponible: https://www.ngmn.org/work-programme/5g-white-paper.html

[9] H. Biallach, M. Bouhtou et D. Nace, "Optimization of the virtual network function reconfiguration plan in 5g network slicing," *ICN International Conference on Networks*, p. 28–34, 2022.

[10] ——, "An efficient heuristic for the virtual network function reconfiguration problem," *12th International Workshop on Resilient Networks Design and Modeling (RNDM)*, 2022, (to appear).

[11] K. Kumbria *et al.*, "Vnfs reconfiguration in 5g networks," *12th International Workshop on Resilient Networks Design and Modeling (RNDM)*, 2022, (to appear).

[12] H. Mehta *et al.*, "0g to 5g mobile technology: A survey," 09 2014.

[13] N. Alliance, "5g white paper," *Next generation mobile networks, white paper*, vol. 1, 2015.

[14] R. Mijumbi *et al.*, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys Tutorials*, vol. 18, nº. 1, p. 236–262, 2016.

[15] B. Yi *et al.*, "A comprehensive survey of network function virtualization," *Computer Networks*, vol. 133, p. 212–262, 2018. [En ligne]. Disponible: https://www.sciencedirect.com/science/article/pii/S1389128618300306

[16] ETSI, "Etsi, network functions virtualisation - introductory white paper," p. 1–16, 2012. [En ligne]. Disponible: https://portal.etsi.org/NFV/NFV_White_Paper.pdf

[17] D. Cho *et al.*, "Real-time virtual network function (vnf) migration toward low network latency in cloud environments," *IEEE 10th International Conference on Cloud Computing (CLOUD)*, p. 798–801, 2017.

[18] A. Gausseran *et al.*, "Be scalable and rescue my slices during reconfiguration," *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, p. 1–6, 2020.

[19] Y. Liu *et al.*, "On dynamic service function chain reconfiguration in iot networks," *IEEE Internet of Things Journal*, p. 1–1, 2020.

[20] S. Sotiriadis *et al.*, "Vertical and horizontal elasticity for dynamic virtual machine reconfiguration," *IEEE Transactions on Services Computing*, p. 1–1, 2016.

[21] O. Houidi *et al.*, "An efficient algorithm for virtual network function scaling," *GLOBECOM, IEEE Global Communications Conference*, p. 1–7, 2017.

[22] P. T. A. Quang *et al.*, "Single and multi-domain adaptive allocation algorithms for vnf forwarding graph embedding," *IEEE Transactions on Network and Service Management*, vol. 16, n°. 1, p. 98–112, 2019.

[23] O. Soualah *et al.*, "A green vnf-fg embedding algorithm," *4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, p. 141–149, 2018.

[24] J. J. A. Esteves *et al.*, "Location-based data model for optimized network slice placement," *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, p. 404–412, 2020.

[25] A. Laghrissi et T. Taleb, "A survey on the placement of virtual resources and virtual network functions," *IEEE Communications Surveys Tutorials*, vol. 21, n°. 2, p. 1409–1434, 2019.

[26] X. Li et C. Qian, "A survey of network function placement," *13th IEEE Annual Consumer Communications Networking Conference (CCNC)*, p. 948–953, 2016.

[27] V. Medina et J. M. García, "A survey of migration mechanisms of virtual machines," *ACM Comput. Surv.*, vol. 46, n°. 3, janv. 2014. [En ligne]. Disponible: https://doi.org/10.1145/2492705

[28] D. Kapil, E. S. Pilli et R. C. Joshi, "Live virtual machine migration techniques: Survey and research challenges," dans *2013 3rd IEEE International Advance Computing Conference (IACC)*, 2013, p. 963–969.

[29] R. Sirdey *et al.*, "On a resource-constrained scheduling problem with application to distributed systems reconfiguration," *European Journal of Operational Research*, vol. 183, p. 546–563, 2007.

[30] V. Kherbache, Madelaine et F. Hermenier, "Scheduling live-migrations for fast, adaptable and energy-efficient relocation operations," *IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, p. 205–216, 2015.

[31] V. Kherbache, E. Madelaine et F. Hermenier, "Memory and Network Aware Scheduling of Virtual Machine Migrations," *EuroSys'15*, 2015, poster. [En ligne]. Disponible: https://hal.inria.fr/hal-01271665

[32] V. Eramo, M. Ammar et F. G. Lavacca, "Migration energy aware reconfigurations of virtual network function instances in nfv architectures," *IEEE Access*, vol. 5, p. 4927–4938, 2017.

[33] V. Eramo *et al.*, "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, n°. 4, p. 2008–2025, 2017.

[34] A. Gausseran *et al.*, "No interruption when reconfiguring my sfcs," *2019 IEEE 8th International Conference on Cloud Networking (CloudNet)*, p. 1–6, 2019.

[35] J. Liu *et al.*, "On dynamic service function chain deployment and readjustment," *IEEE Transactions on Network and Service Management*, vol. 14, n°. 3, p. 543–553, 2017.

[36] O. A. Wahab *et al.*, "Maple: A machine learning approach for efficient placement and adjustment of virtual network functions," *Journal of Network and Computer Applications*, vol. 142, p. 37–50, 2019. [En ligne]. Disponible: https://www.sciencedirect.com/science/article/pii/S1084804519301924

[37] W. Chen *et al.*, "A two-level virtual machine self-reconfiguration mechanism for the cloud computing platforms," *9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing*, p. 563–570, 2012.

[38] M. Pozza *et al.*, "On reconfiguring 5g network slices," *IEEE Journal on Selected Areas in Communications*, vol. 38, n°. 7, p. 1542–1554, 2020.

[39] F. Hermenier *et al.*, "Entropy: a consolidation manager for clusters," *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, 2009.

[40] Y. Liu *et al.*, "An approach for service function chain reconfiguration in network function virtualization architectures," *IEEE Access*, vol. 7, p. 147 224–147 237, 2019.

[41] T. K. Sarker et M. Tang, "Performance-driven live migration of multiple virtual machines in datacenters," *IEEE International Conference on Granular Computing (GrC)*, p. 253–258, 2013.

[42] R. Mijumbi *et al.*, "Design and evaluation of algorithms for mapping and scheduling of virtual network functions," *Proceedings of the 1st IEEE Conference on Network Softwarization (NetSoft)*, p. 1–9, 2015.

[43] I. Fajjari *et al.*, "Vnr algorithm: A greedy approach for virtual networks reconfigurations," *IEEE Global Telecommunications Conference - GLOBECOM 2011*, p. 1–6, 2011.

[44] S. Ayoubi, Y. Zhang et C. Assi, "A reliable embedding framework for elastic virtualized services in the cloud," *IEEE Transactions on Network and Service Management*, vol. 13, n°. 3, p. 489–503, 2016.

[45] H. Jin *et al.*, "Dynamic processor resource configuration in virtualized environments," *IEEE International Conference on Services Computing*, p. 32–39, 2011.

[46] P. Campegiani *et al.*, "A genetic algorithm to solve the virtual machines resources allocation problem in multi-tier distributed systems," *Second International Workshop on Virtualization Performance: Analysis, Characterization, and Tools (VPACT 2009), Boston, Massachusett*, 2009.

[47] H. Mi *et al.*, "Online self-reconfiguration with performance guarantee for energy-efficient large-scale cloud computing data centers," *Proceedings - IEEE 7th International Conference on Services Computing, SCC 2010*, p. 514 – 521, 2010.

[48] S. Kim, Y. Han et S. Park, "An energy-aware service function chaining and reconfiguration algorithm in nfv," *IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, p. 54–59, 2016.

[49] T. Hirayama *et al.*, "Service function migration scheduling based on encoder-decoder recurrent neural network," *IEEE Conference on Network Softwarization (NetSoft)*, p. 193–197, 2019.

[50] F. Wei *et al.*, "Network slice reconfiguration by exploiting deep reinforcement learning with large action space," *IEEE Transactions on Network and Service Management*, p. 1–1, 2020.

[51] X. Guo *et al.*, "Qos-aware data center network reconfiguration method based on deep reinforcement learning," *Journal of Optical Communications and Networking*, vol. 13, 2021.

[52] N. Jussien, G. Rochart et X. Lorca, "Choco: an open source java constraint programming library," *CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08)*, 2008.

[53] O. Stan *et al.*, "A GRASP for placement and routing of dataflow process networks on many-core architectures," *Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2013, Compiegne, France, October 28-30, 2013*, p. 219–226, 2013.

[54] G. Forney, "The viterbi algorithm," *Proceedings of the IEEE*, vol. 61, n°. 3, p. 268–278, 1973.

[55] D. Yu *et al.*, "An improved k-medoids algorithm based on step increasing and optimizing medoids," *Expert Systems with Applications*, vol. 92, 2017.

[56] S. Padhy et J. Chou, "Reconfiguration aware orchestration for network function virtualization with time-varied workload in virtualized datacenters," *IEEE Access*, p. 1–1, 2021.

[57] G. Wang *et al.*, "Reconfiguration in network slicing - optimizing the profit and performance," *IEEE Transactions on Network and Service Management*, 2019.

[58] G. Chartrand et P. Zhang, *A first course in graph theory.* Courier Corporation, 2013.

[59] A. B. Kahn, "Topological sorting of large networks," *Commun. ACM*, vol. 5, n°. 11, p. 558–562, nov. 1962. [En ligne]. Disponible: https://doi.org/10.1145/368996.369025

[60] D. C. Kozen, "Depth-first and breadth-first search," *The design and analysis of algorithms*, p. 19–24, 1992.

[61] A. Baharev *et al.*, "An exact method for the minimum feedback arc set problem," *ACM J. Exp. Algorithmics*, vol. 26, avr. 2021.

[62] R. M. Karp, "Reducibility among combinatorial problems," *Complexity of computer computations*, p. 85–103, 1972.

[63] B. Schwikowski et E. Speckenmeyer, "On enumerating all minimal solutions of feedback problems," *Discrete Applied Mathematics*, vol. 117, n°. 1-3, p. 253–265, 2002.

[64] P. Eades, X. Lin et W. F. Smyth, "A fast and effective heuristic for the feedback arc set problem," *Information Processing Letters*, vol. 47, n°. 6, p. 319–323, 1993.

[65] P. Eades et X. Lin, "A heuristic for the feedback arc set problem." *Australas. J Comb.*, vol. 12, p. 15–26, 1995.

[66] F. J. Brandenburg et K. Hanauer, "Sorting heuristics for the feedback arc set problem," *Technical report, Technical Report MIP-1104*, 2011.

[67] G. B. Dantzig, "Linear programming," *Operations research*, vol. 50, n°. 1, p. 42–47, 2002.

[68] H. M. Salkin et C. A. De Kluyver, "The knapsack problem: a survey," *Naval Research Logistics Quarterly*, vol. 22, n°. 1, p. 127–144, 1975.

[69] D. E. Knuth et J. L. Szwarcfiter, "A structured program to generate all topological sorting arrangements," *Information Processing Letters*, vol. 2, n°. 6, p. 153–157, 1974.

[70] D. Ajwani, A. Cosgaya-Lozano et N. Zeh, "A topological sorting algorithm for large graphs," *ACM J. Exp. Algorithmics*, vol. 17, 2012.

[71] J. Sun *et al.*, "Breaking cycles in noisy hierarchies," *Proceedings of the 2017 ACM on Web Science Conference*, p. 151–160, 2017. [En ligne]. Disponible: https://doi.org/10.1145/3091478.3091495

[72] Breaking cycles in noisy hierarchies. [En ligne]. Disponible: https://github.com/zhenv5/breaking_cycles_in-noisy_hierarchies

[73] R. Liu, "A low complexity topological sorting algorithm for directed acyclic graph," vol. 4, p. 194–197, 2014. [En ligne]. Disponible: http://www.ijmlc.org/papers/411-LC039.pdf

[74] T. Ahammad, M. Hasan et M. Zahid Hassan, "A new topological sorting algorithm with reduced time complexity," *Intelligent Computing and Optimization*, p. 418–429, 2021.

[75] J. Zhou et M. Müller, "Depth-first discovery algorithm for incremental topological sorting of directed acyclic graphs," *Inf. Process. Lett.*, vol. 88, n°. 4, p. 195–200, nov. 2003.

[76] M. C. Er, "A parallel computation approach to topological sorting," *THE COMPUTER JOURNAL*, vol. 26, 1983.

[77] A. Somani et D. P. Singh, "Parallel genetic algorithm for solving job-shop scheduling problem using topological sort," *International Conference on Advances in Engineering Technology Research (ICAETR - 2014)*, p. 1–8, 2014.

[78] F. Solano et M. Pióro, "Lightpath reconfiguration in wdm networks," *Journal of Optical Communications and Networking*, vol. 2, n°. 12, p. 1010–1021, 2010.

[79] R. S. Pereira *et al.*, "Reliable: Resource allocation mechanism for 5g network using mobile edge computing," *Sensors*, vol. 20, n°. 19, p. 5449, 2020.

[80] E. R. Sanseverino *et al.*, "Dynamic programming and munkres algorithm for optimal photovoltaic arrays reconfiguration," *Solar Energy*, vol. 122, p. 347–358, 2015.

[81] Y. Song, C. Zhang et Y. Fang, "Multiple multidimensional knapsack problem and its applications in cognitive radio networks," *MILCOM IEEE Military Communications Conference*, p. 1–7, 2008.

[82] R. Sirdey, J. Carlier et D. Nace, "Approximate solution of a resource-constrained scheduling problem," *J. Heuristics*, vol. 15, n°. 1, p. 1–17, 2009.

[83] T. Cormen, C. Leiserson et R. Rivest, "Dynamic programming, introduction to algorithms," 1990.

[84] T. H. Cormen *et al.*, "Dynamic programming," *Algorithms for Parallel Processing*, vol. 105, p. 307, 1998.