Modeling and optimization of 5G network design

Wesley da Silva Coelho

HESAM UNIVERSITÉ

le cnam

ÉCOLE DOCTORALE SCIENCES DES MÉTIERS DE L'INGÉNIEUR
Centre d'études et de recherche en informatique et communications

THÈSE

présentée par : **Wesley DA SILVA COELHO**
soutenue le : **8 novembre 2021**

pour obtenir le grade de : **Docteur d'HESAM Université**

préparée au : **Conservatoire national des arts et métiers**

Discipline : **Informatique**

Spécialité : **Informatique**

Modeling and Optimization of 5G Network Design

THÈSE dirigée par :
M. SECCI Stefano, Professeur, CNAM

Jury

Safia KEDAD-SIDHOUM	CNAM	Président
Bernardetta ADDIS	Université de Lorraine	Rapporteuse
Pierre FOUILHOUX	Paris Nord	Rapporteur
Tijani CHAHED	Telecom SudParis	Examinateur
Bernard FORTZ	Université Libre de Bruxelles	Examinateur
Stefano SECCI	CNAM	Directeur de thèse
Amal BENHAMICHE	Orange	Co-encadrente
Nancy PERROT	Orange	Co-encadrente

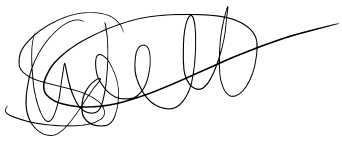
THÈSE

Affidavit

I, undersigned, Wesley da Silva Coelho, hereby declare that the work presented in this manuscript is my own work, carried out under the scientific direction of Stefano Secci (thesis director), in accordance with the principles of honesty, integrity and responsibility inherent to the research mission. The research work and the writing of this manuscript have been carried out in compliance with the French charter for Research Integrity. This work has not been submitted previously either in France or abroad in the same or in a similar version to any other examination body.

Paris, November 8th 2021

Signature



Résumé

Chaque seconde, une grande quantité de données numériques est transportée à travers une infinité de types d'appareils via les réseaux cellulaires du monde entier, et les attentes sont à une croissance considérablement accélérée avec des demandes de plus en plus importantes. En quelques années, ces réseaux pourraient ainsi atteindre leurs capacités maximales en termes de transmission de données. Pour faire face ces défis, Network Slicing a été présenté comme une nouvelle infrastructure virtualisée pour le système de réseau mobile de nouvelle génération. Cette technologie couvre désormais non seulement le niveau des applications, mais également la virtualisation des couches physiques et de commutation, avec différentes technologies d'accès radio. Ainsi, chaque fournisseur de services doit pouvoir déployer ses services et applications sur des réseaux logiques, appelés Network Slices, spécifiquement adaptés à ses exigences techniques. Le mode de communication Device-to-Device est une autre approche présentée comme une alternative prometteuse à la communication traditionnelle dans les réseaux cellulaires. Cette technologie permet de réutiliser les ressources radio et de diminuer la latence de bout en bout des communications locales. Par conséquent, l'optimisation des ressources physiques dans les réseaux cellulaires devient cruciale pour mieux dimensionner et déployer les réseaux virtuels. L'objectif global de ce travail est donc de définir et d'étudier le concept de Device-to-Device Communication et Network Slice Design dans les systèmes 5G, en proposant des modèles mathématiques et des algorithmes innovants pour résoudre les problèmes d'optimisation sous-jacents. Mots-clés : Optimisation, Conception de réseau, Découpage réseau en tranches, Communication d'appareil à appareil.

Abstract

Every second, a large amount of digital data is transported through huge number of types of devices via cellular networks worldwide, and expectations are at a greatly accelerated growth with increasingly large requests. In few years, these networks could thereby reach their maximum capacities in terms of data transmission. To face these challenges, Network Slicing has been presented as a novel virtualized infrastructure for the new generation cellular network system. This technology now not only covers application-level abstraction but also physical and switching layers virtualization, with different radio access and link communication technologies. Hence, each service provider is to be able to deploy its communication services on top of logical networks, named Network Slices, specifically tailored to its technical requirements. The Device-to-Device communication mode is another approach presented as a promising alternative to traditional communication in cellular networks. This technology allows to reuse radio resources and to decrease the end-to-end latency of local communications. Consequently, the optimization of physical resources in cellular networks becomes crucial to better deploy virtual networks. The overall objective of this work is therefore to define and study the concept of device-to-device communication and network slice design in 5G systems, and propose mathematical models and innovative algorithms to solve the underlying optimization problems.

Keywords : Optimization, Network Design, Network Slicing, Device-to-Device.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The expectations around the data flow on mobile networks are at a greatly accelerated growth with increasingly large requests. In few years, these networks could consequently reach their maximum capacities in terms of data transmission. To face all these challenges, 5G technology is posed by enabling the digitization of society and economic information. The idea behind the concept of 5G is that it does not correspond to a simple increase in data rate, as was the case for previous generations, but also the aim is thereby to widen the diversity of user equipment (UE).

This technological evolution will touch the whole network environment, going from cellular and radio access to the application service architectures. This transition challenges network design since multiple resources and segments, historically managed independently, are to be operated with both continuity in networking and computing resource allocation and provisioning an as-a-whole and unique service. In this context, different providers can be associated with different communication services (CS) running on the same physical network at the access, core, and application segments.

Such communication services can be of three classes: enhanced Mobile Broadband (eMBB; e.g., broadband everywhere and large-scale events), Ultra-Reliable Low Latency Communications (URLLC; e.g., online gaming and autonomous driving), and massive Machine Type Communications (mMTC; e.g., device-to-device communication and internet of things) - differing in the requirements, such as maximum latency, minimum availability, and bandwidth. To provide the necessary flexible provisioning, Network Function Virtualization [2], Software Defined Networking [3], and Network Slicing [4] technologies can be adopted to let the CS provider deploy its services on top of logical networks.

Because of different bitrate and latency requirements, policies on radio access function splitting

are also going to have an impact on the backhauling network dimensioning, and therefore on the placement of core network functions and on the configuration of edge computing application servers. Moreover, different policies for control versus data-plane function sharing and scaling are expected to be applied.

To overcome these challenges, 'Network Slice' has been presented as a novel virtualized infrastructure model. This technology not only covers the application-level abstraction but also the physical and switching layers virtualization, with different radio access and link communication technologies. Hence, each service provider is to be able to deploy its communication services on top of logical networks, named Network Slices, specifically tailored to its technical requirements.

The Device-to-Device (D2D) communication mode is another new approach presented as a promising alternative to traditional communication in cellular networks. This technology allows to reuse radio resources and to decrease the end-to-end latency of local communications. Then, D2D would allow a set of UEs geographically close to each other to establish direct D2D communications, or span multiple links (multi-hop D2D communications), to access a given service (e.g. video streaming or gaming) while ensuring the required service quality.

In this context, optimizing resources in cellular networks becomes crucial on backhauling network dimensioning, and hence on the placement of core network functions and the configuration of edge computing application servers. Moreover, different policies for control versus data-plane function sharing and scaling are expected to be applied. All these challenges must be overcome wisely and effectively, as the state of the system can change every second. In legacy technologies, such as 3G and 4G, the entire network system was designed for approximately ten years of use, with small variations and slow evolution over the period. Contrarily, the 5G system is meant to be extremely flexible and still be able to offer a customized and complete virtual network in a few minutes for each Communication Service request. The overall objective of this work is therefore to define and study the concept of device-to-device and network slice design in 5G systems and propose mathematical models and innovative algorithms to solve the underlying optimization problems.

This manuscript is organized as follows. We first briefly present the basic notions on graph theory and combinatorial optimization in Chapter 2. Then, in Chapter 3, we overview the evolution of the mobile system, from 3G to 5G, and present the entities appearing in new generation networks as well the main modeling aspects and technical constraints related to 5G systems and beyond. The content

presented in Chapters 3 was partially published in the IEEE Communications Standards Magazine [1].

In Chapter 4, we formally define the Domain Creation Problem (DCP) and we propose a node-arc (compact) integer linear programming (ILP) formulation to model it. We then present some strategies to enhance the linear relaxation of the formulation along with two classes of valid inequalities within a branch-and-cut algorithm to solve the problem. We also propose another solving method to address the DCP, which is based on a decomposition of the problem into two sub-problems: the *routing sub-problem* and the *resource allocation sub-problem*, that are to be solved separately. We further propose a two-phase heuristic, obtained by such decomposing. To solve the routing sub-problem, we propose two methods: an LP-based heuristic from the linear relaxation of a compact formulation, and a non-compact formulation obtained by generating a subset of relevant paths. Then, the allocation sub-problem is transformed into a vertex coloring problem that is solved heuristically by an improved greedy algorithm. The content of Chapter 4 was presented at the International Network Optimization Conference and published in its proceedings [5].

In Chapter 5, we introduce and study the Network Slice Design Problem (NSDP) in 5G systems. We first propose a mixed-integer linear programming (MILP) formulation for the problem including novel *splitting*, *mapping* and *provisioning* constraints described in the published 5G standards documents [6, 7, 8]. Then, we model new variants and extensions of the problem: NDSP with intra-slice flexible splitting (NSDP-ISFS) and NDSP with inter-slice split continuity. We then provide several sensitivity analyses regarding the impact of each proposed variant on the network. In Chapter 6, we propose several classes of valid inequalities in order to strengthen the linear relaxation of the proposed MILP and integrate them in a Branch-and-Cut framework to solve the problem. We further present several strategies to reduce the symmetries and the size of the model. We go beyond in Chapter 7 by proposing an open-access framework based on a math-heuristic for the Network Slice Design that relies on decomposing the NSDP into a few sub-problems and sequentially solve them. The content presented in both Chapters 5 and 7 was respectively presented at the International Conference on Network and Service Management and at the International Teletraffic Congress, and published in their related proceedings [9, 10].

We dedicate Chapter 8 to present another variant of the NSDP, where dedicated network functions are deployed to each network slice. Different strategies are then proposed in order to efficiently solve the optimization problems related to the problem. First, we propose a compact formulation and an

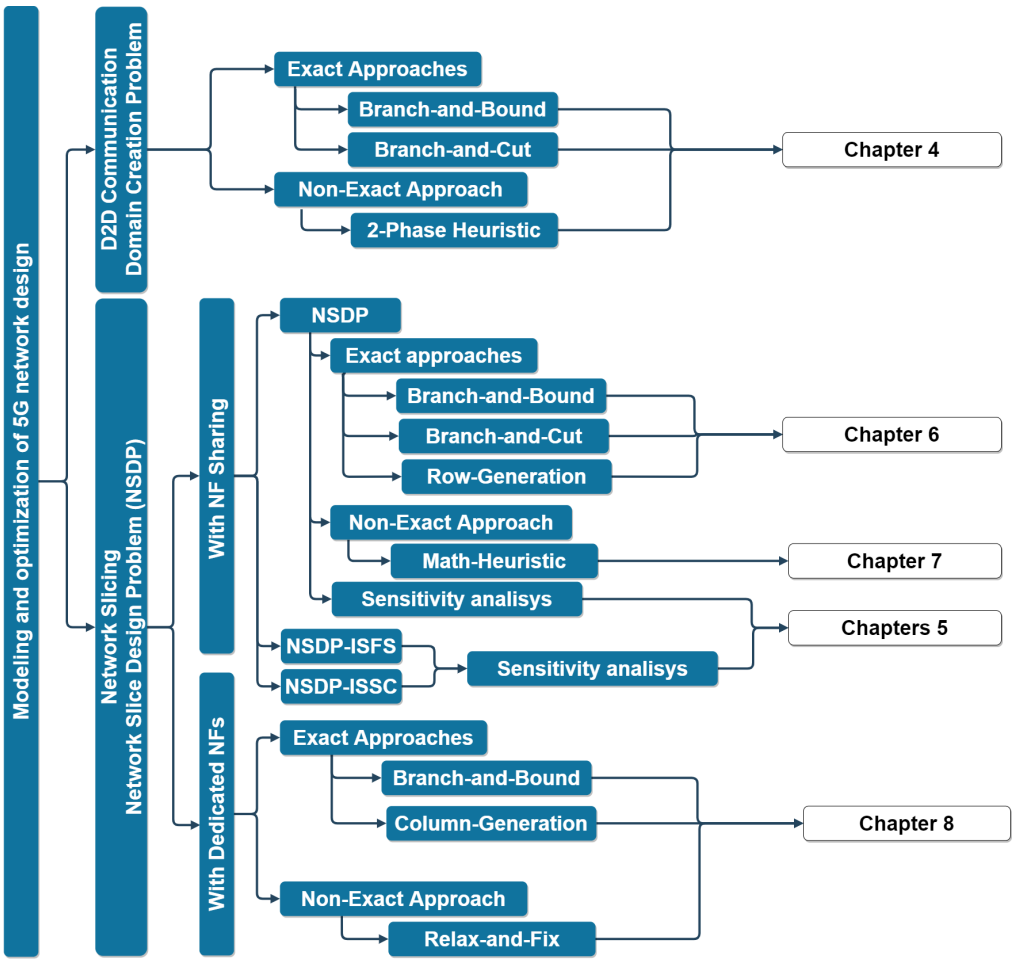


Figure 1.1 – Contributions and manuscript organization.

extended one. To solve the former, we proposed a Relax-and-Fix heuristic that relies on repetitively solving the proposed related ILP with only a few integer variables and fixing or relaxing most of the remaining integer and binary ones. On the other hand, to address the exponential number of variables in the extended formulation, a column generation-based framework is then proposed.

Figure 1.1 summarizes our contributions and the organization of this manuscript.

Chapter 2

Background

Contenu

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 and non-exact approaches commonly used to address optimization problems.

2.1 Theory of graphs

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 cellular networks.

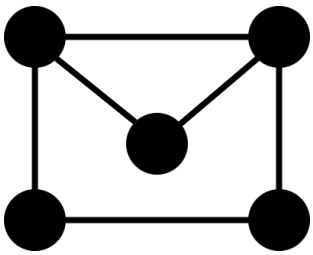


Figure 2.1 – Representation of a graph

2.1.1 Definitions

As described in [11], a graph is composed of vertices and arcs (or edges). A graph G is defined formally by a pair (V, E) where V is a finite set of elements called a *vertex*, and a subset (possibly empty) E is of pairs of elements of V. Each element of the subset E is called *arc* if the graph is oriented - or *edge* if the graph is a non-oriented graph.

The set E of an oriented graph, for example, is therefore composed of pairs (u, v), where u and v are called *origin* and *destination* respectively. We call *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.

Non-oriented graphs are graphs that have no orientation on their arcs (we, therefore, use the term *edge*), which means that if there is an edge between the vertex u and v, the information flow can be sent in both directions: from u to v and also from v to u. Both u and v can be the origin or destination of the edge at a time, depending on the case. Figure 2.1 shows a representation of a non-oriented graph of size 6 and order 5.

In this context, a *path* of length k between a vertex u and a vertex v of a graph $G = (V, E)$ is a vertices sequence $v_0, v_1, ..., v_k$ such that $u = v_0, v = v_k$, and (v_{i-1}, v_i) belongs to E for $i = 1, ..., k$. We say that a vertex v is *reachable* by a vertex u if there is a path between u and v such that its length k is greater than zero. Two vertices are called *adjacent* if $k = 1$. In this context, a *complete graph* is a graph where all vertices are adjacent to each other, that means that every pair of disjoint vertices is connected by an edge. A *cycle* in the graph is defined as a path $v_0, v_1, ..., v_k$ such that $k > 0$ and $v_0 = v_k$. Finally, a path is called *elementary* if all its vertices are visited only once.

2.1.2 Some classical problems

We dedicate this sub-section to make a brief introduction to some important problems involving graphs that were used in this work.

2.1.2.1 Path finding problem

Pathfinding is a problem that consists in finding how to move in an environment between a starting point and a point of arrival taking into account different constraints. This problem is intensively studied and used in various fields such as transportation [12, 13, 14, 15] and telecommunications [16, 17].

However, pathfinding becomes a complex problem when attempting to take into account various additional constraints (E.g., real-time execution, presence of uncertainties, resource constraints, and scalable environment). Initially, a path-finding problem can be reduced to a problem of finding the shortest path between two nodes in a graph. There are several ways to solve the shortest path problems, each one is adapted to a set of different problems [18]:

— Problem with a single destination: consists of determining the shortest path between each vertex of the graph and a given destination vertex;

— Problem with a single origin: determines the shortest path between a given vertex and all the other vertex of the graph;

— Origin-destination problem: determines the shortest path between two vertices given;

— All shortest paths problem: determines the shortest path between each pair of vertices present in the graph.

According to the same authors, specific algorithms to solve the shortest path problem are sometimes called *path search algorithms*. Among the algorithms of this class, the most known are:

— Dijkstra's algorithm [19]: it solves the problem with an origin vertex in a graph whose edges have weights greater than or equal to zero. This algorithm can determine the shortest path to all the other vertices of the graph starting from a given vertex;

— Bellman-Ford's algorithm [20]: it solves the problem with an original vertex in a graph whose edges can have negative weights;

— Johnson's algorithm [21]: it determines the distance between all vertices pairs in a graph.

— Breadth-First Search (BFS) [22]: it calculates the distances of all nodes from a source node in

an unweighted graph (oriented or non-oriented);

— Yen's algorithm [23]: computes single-source K-shortest loopless paths for a graph with non-negative edge weight.

In telecommunications and signal processing, the modeling might be slightly different (probabilistic), and the problem can be solved by the Viterbi's algorithm [24].

2.1.2.2 Max-clique problem

A clique in a graph $G = (V, E)$ is a sub-set of vertices $C \subseteq V$, such that for every two vertices in C, there is an edge connecting them. This is equivalent to saying that the induced sub-graph of C is complete.

It is important to differentiate a *maximal clique* from a *maximum clique*. A maximal clique is a clique that can not be extended by adding one or more adjacent vertices. On the other hand, a maximum clique is the largest possible clique on a given graph. The clique number $\omega(G)$ of a graph G is the number of vertices of its maximum clique.

The optimization problem associated with the "clique problem" is the problem of the maximum clique: it consists in finding the largest clique (in the sense of its number of vertices) in a graph. The search for a clique of maximal size in a graph is a classical problem of the complexity theory. The maximum clique problem is one of the 21 NP-complete problems of Karp published in 1972 in Reducibility Among Combinatorial Problems [25].

2.1.2.3 Coloring problem

In graph theory, graph coloring is a special case of graph labeling. It is an assignment of labels traditionally called "colors". In its simple form, it is a process where a different color tone is assigned to each vertex of a graph, such that there is no adjacent vertex sharing the same color. This is called *vertex coloring*. Likewise, the *edges coloring* assigns a color to each edge so that there are no two adjacent edges of the same color.

The *graph coloring problem* can be formulated in two ways. First, as a decision problem, in which the objective is to identify if a graph is colorable with exactly k colors. In the second formulation, which is related to optimization problems, the objective is to find the minimum value of k. The fact that this problem is classified as NP-hard constitutes a shred of strong evidence that there are no

polynomial algorithms that can be used in its resolution, justifying therefore the use of enumeration methods or development and use of heuristics [26]. The size of the maximum clique of a graph is the best-known lower bound for the vertex coloring problem.

2.2 Linear programming

The development of linear programming has been classified as one of the greatest scientific advances since the 1950s. According to [27], the problem of optimizing a linear function subject to linear constraints originated with the Fourier studies of linear inequalities systems in 1826. However, only in 1939, Kantorovich [28] notes the practical importance of these questions, creating an algorithm for its solution.

This type of problem peaked with George Dantzig in the 1940s, with the formulation of the diet problem [29] as a problem of mixing components. Danzig not only formulates the problem of linear programming but also creates the Simplex algorithm for its solution in 1947. Its impact on the mid-twentieth century has allowed companies to save millions of euros and continues to help the various industrial and service sectors nowadays.

According to [30], a general problem to be solved by linear programming can be defined as the objective of allocating in the best possible way - *optimal* - limited resources for activities that compete with one another. Thus, the choice of the level of activity determines the quantity of each resource consumed at each activity.

Linear programming, therefore, uses a mathematical model to describe the problem studied. According to [27], the adjective *linear* means that all the mathematical functions of this model are necessarily linear functions[1]. On the other hand, the noun *programming*, in this context, does not refer directly to computer programming. It is, in essence, a synonym for *planning*.

Hence, linear programming winds up the planning of activities to obtain an optimal result, that is a result that makes it possible to obtain the best allocation of resources between all the possible scenarios. We can therefore define a generic model for any linear programming problem as follows:

$$max\ c^t x \tag{2.1}$$

1. An affine function is a function of the type $f(x) = ax + b$, where x is variable and a and b are real constants. Its graph is always a straight line. A linear function, however, is an affine function where b is equal to zero, taking the form $f(x) = ax$.

$$Ax \leq b \tag{2.2}$$

$$x \geq 0 \tag{2.3}$$

where c and b are vectors with constant values and have the size of the decision variables and the number of constraints, respectively. A is a matrix $[|b|, |c|]$ also having constant values to represent the coefficients of each variable in each constraint. Here, x is the representation, also by a vector, of the decision variables. The Simplex algorithm is a powerful tool for solving this type of problem and we discuss it in the following sections.

2.2.1 Integer linear programming

A *integer linear programming* problem is a linear programming problem in which all of the variables are discrete - they must be integer values [27]. Similarly, if there are continuous and discrete variables, then the mathematical model is a problem of a *mixed integer linear programming* (MILP).

We can therefore write the program as follows:

$$min\{cx : Ax \leq b, x \geq 0, x \in \mathbb{Z}\}$$

The author in [31] considers the problems of linear programming involving *yes or no* decisions as one of its most important applications. In these decisions, the only possible choices are *yes* or *no*. With only two options, these decisions can be represented by decision variables that are limited to only two values, typically 0 or 1.

Consequently, the i-th decision of this nature is represented by x_i, such that:

$$x_i = \begin{cases} 1 & \text{, if the decision is } yes \\ 0 & \text{, if the decision is } no \end{cases}$$

These variables are called *binary variables*. Therefore, linear programming problems that contain binary variables can be classified as *binary integer programming*.

2.2.2 Geometric aspects of linear programming

In this subsection, we focus on the geometric aspects of Linear Programming, introducing the concepts of *Convexity*, *Polyhedron* and *Facets*. First, a point x is a convex combination of m points

in $x_j R^n, \forall j = 1, .., m$, if and only if

$$x = \sum_{j=1}^{m} \lambda_j x_j$$
$$\sum_{j=1}^{m} \lambda_j = 1$$
$$\lambda_j \geq 0 \quad , \forall j = 1..m \qquad bibitem_R AC$$

In particular, a point $x(\lambda)$ is a convex combination of two points z and w, if

$$x(\lambda) = (1 - \lambda)z + \lambda w,\ \lambda \in [0, 1]$$

Geometrically, they are the points on the straight line that goes from z to w. A set $C \in R^n$ is said to be convex when, for all x and y of C, the interval $[x, y]$ is contained in C. That is to say:

$$\lambda x + (1 - \lambda)y \in C\ ,\ \forall x, y \in C, \forall \lambda \in [0, 1]$$

We call *extreme point* of a convex set C any point x which can not be represented as a convex combination of two distinct points of C. The C profile is the set of all extreme points of C. We call *convex envelope* of a set C, the intersection of all (closed) convex sets containing C. A convex and compact - closed and bounded - set C is equal to the closed convex envelope of its profile.

A constraint in R^n is an inequality of the form

$$g(x) \leq 0,\ g :\ R^n \to R$$

and a feasible region defined by m constraints of the type $g(x)$ is the set of feasible x points. A polyhedron in R^n is a feasibility region defined by linear constraints:

$$Ax \leq d$$

We say that in a feasible point x, the constraint $g_i(x) \in A$ is active if and only if it is of the form $g_i(x) = d$. It is worthwhile mentioning that polyhedrons are convex, but not necessarily limited.

2.2.3 Duality in linear programming

According to [27], any linear programming problem is associated with another linear programming problem called *dual*. And this original problem is called the *primal problem*. The relationship between the dual problem and the original problem (primal) is extremely useful.

Let a primal linear programming problem be described in the following way on the left, and for its corresponding dual problem we have the model on the right:

$PRIMAL$	$DUAL$
$max\ c^t x$	$min\ b^t y$
$Ax \leq b$	$A^t y \geq c$
$x \geq 0$	$y \geq 0$

If the primal problem is maximization, the dual problem is minimization. The dual problem uses the same parameters as its associated primal problem, but in the following way:

— The coefficients of the objective function of the primal problem are the right sides of the constraints in the dual problem.

— The right sides of the constraints in the primal problem are the coefficients of the objective function of the dual problem.

— The coefficients of a variable in the constraints of the primal problem are the coefficients of a constraint of the dual problem.

Knowing that the dual is also a problem of linear programming, one can say that there exists another associated problem of linear programming called dual. If one goes back to the problem, one sees that the latter is none other than the dual problem of departure. Thus we have that the dual of a dual problem is equal to the primal problem.

The main results of applying the primal-dual relation can be summarized in the following theorems, whose proofs can be found in [32, 33]:

The weak duality theorem: If x is a feasible solution of the primal, and y a feasible dual solution, then necessarily $c^t x \leq b^t y$. In particular, if $c^t x = b^t y$ then x is an optimal solution of the primal and y is an optimal solution of the dual.

The strong duality theorem: If the primal problem has an optimal solution x then the dual problem has an optimal solution y. In this case, we necessarily have $c^t x = b^t y$.

Complementary gaps theorem: If x is a feasible solution of the primal problem, and y a feasible solution of the dual problem, then x and y are optimal feasible solutions if and only if the following conditions are satisfied:

— If a constraint is satisfied as a strict inequality in the primal (dual) then the corresponding variable of the dual (primal) is zero.

— If the value of a variable in the primal (dual) is strictly positive then the corresponding constraint of the dual (primal) is an equality.

The dual problem also has a much-used economic interpretation, called *reduced cost*. The value of the dual variable corresponding to a constraint of the primal gives the value of the reduced cost of this constraint. We consider the reduced cost as the potential gain we can have in the objective function of the primal: its value gives us the effect on the objective function if we increase by one the right side of the constraint. We can compute the reduced cost cr associated with a variable x of the primal by: $cr = c - A^t y$. If we look at the constraint $A^t y \geq c$ of the dual model described above, we can easily see that the reduced cost is associated with the fact that the constraint is satisfied or not.

2.3 Solving approaches for linear programming

In this section, we describe the most efficient methods for solving linear programs. We first introduce the Simplex method. Then, we cover the methods of linear relaxation and cutting planes. We also introduce the basis of efficient branch-and-bound and column generation methods. Finally, we present some non-exact methods such as heuristics and meta-heuristics.

2.3.1 Graphical method and Simplex

The *Graphical method* - or *Geometric method* - allows solving simple linear programming problems in an intuitive and visual way. This method is limited to problems with two or three decision variables since it is not possible to graphically illustrate more than 3 dimensions. According to [34], this method is a visual aid to interpret and understand the algorithm of the Simplex method (much more sophisticated and abstract) and the concepts that surround it. The phases of the problem-solving process using the Graphic method are as follows:

— Draw a Cartesian coordinate system in which each decision variable is represented by an axis.

— Establish a scale of measurement for each of these axes appropriate to the associated variable.

— Draw the constraint coordinates of the problem, including non-negative (which will be the axes themselves). Note that an inequality defines a region that will be the half-plane bounded by the straight line obtained by considering the constraint as an equality, while an equation defines a region that is the straight line itself.

— The intersection of all regions determines the feasible region or solution space (which is a convex set). If this region is not empty, one must continue to the next step. Otherwise, there is no point that simultaneously satisfies all constraints, so the problem will have no solution and it will be called *unfeasible*.

— Determine the endpoints or vertices of the polygon or polyhedron that form the feasible region. These points will be the candidates for the optimal solution.

— Evaluating the objective function at each vertex. That (or those) that maximize (or minimize) the resulting value will determine the optimal solution to the problem.

The Simplex method is an iterative process that allows improving the solution of the objective function in each step. The process ends when it is not possible to continue improving this value, that is when the optimal solution is obtained. Based on the value of the objective function at any point, the procedure consists in looking for another point that improves the previous value. As can be seen in the graph method, such points are the vertices of the polygon (or polyhedron, if the number of variables is greater than 2). The search is carried out employing displacements by the edges of the polygon, from the current vertex to an adjacent one that improves the value of the objective function. Whenever there is a viable region, and since its number of vertices and edges is finite, it will be possible to find the solution. For problems of maximization, for example, the Simplex method is based on the following property: if the objective function Z does not take its maximum value at vertex A, it means that there is an edge that starts from A and along which the value of Z increases.

2.3.2 Linear relaxation

In integer linear programming, we use the term *relaxation* to indicate that some of the restrictions of the original problem have been relaxed. The most known example is the *linear relaxation*. In this particular case, we remove the integrality constraint on all variables of the original model. Formally, if we have a formulation F represented by

$$z = min\{c(x) : x \in P\}$$

where P is the set of points that satisfy the constraints of the formulation, $c(x)$ is the objective function and z is the value of the optimal solution.

A formulation F_R

$$z_R = min\{f(x) : x \in R\}$$

is a relaxation if all the points of P are in R ($P \subseteq R$) and if $f(x) \leq c(x)$ for $x \in P$. Note that because of these properties, the optimal relaxation solution is a dual limiting for the original formulation, $z_r \leq z$. The dual limiting can be used, for example, to improve the performance of the Branch and Bound algorithm [35] .

The advantage of using relaxations in some cases is that they are easier to solve. For example, linear relaxations can be solved in polynomial time, while the integer version, in general, is much harder to be solved [35]. We can relax a formulation by removing some of its constraints. Note, however, that the more we relax the formulation, the further we will be from the optimal solution. Ideally, one should find a relaxation that is easy to solve and that gives us a good dual limit, that is, very close to the optimal value of the original formulation.

Linear relaxation of an integer linear programming can be solved using a standard linear programming technique, such as Simplex algorithm. If the optimal solution for the relaxed linear program has all the variables equal to an integer value, then it is also the optimal solution for the original integer model [36].

2.3.3 The branch-and-bound framework

To exactly solve an NP-hard optimization problem is often a challenging task requiring very efficient algorithms, and the *Branch and Bound* paradigm is one of the main tools on this. A Branch and Bound algorithm searches the complete solution space for a given problem for the best solution. However, explicit enumeration is normally impossible because of the exponentially increasing number of potential solutions. The use of limits for the function to optimize, combined with the value of the best current solution allows the algorithm to search for parts of the solution space implicitly.

At any time during the process of finding the solution, the state of the solution over the search for the solution space is described by an unexplored subset of this and the best solution found so far. According to [27], at first, a single subset exists, namely the complete solution space, and the best

solution found so far is ∞. Unexplored subspaces are represented as nodes in a dynamically generated search tree, which initially contains only the *root*, and each iteration of the classical algorithm treats such a node. The iteration has three main components: the selection of the node to be processed, calculation of the limit, and the branching step. The sequence of these may vary depending on the strategy chosen to select the next node to be processed.

If the following sub-problem selection is based on the value of the sub-problem boundary, the first operation of an iteration after choosing the node is branching, that is, the subdivision of the space of the solution of the node into two or more subspaces to be studied in a subsequent iteration. For each of them, we check whether the subspace consists of a unique solution, in which case it is compared to the best current solution while keeping the best of them. Otherwise, the subspace selection function is computed and compared to the best current solution. If it can be established that the subspace can not contain the optimal solution, the whole subspace is rejected, otherwise, it is stored in the set of live nodes with its limit. In [37], this is called *impatient strategy* for node evaluation, since the limits are computed as soon as nodes are available.

An alternative is to start with the calculation of the boundary of the selected node, then branch on this node if necessary. The created nodes are then stored with the boundary of the processed node. This strategy is often used when the next node to be processed is chosen to be a living node of the maximum depth in the search tree. The search ends when there are no more unexplored parts of the solution space, and the optimal solution is then the one recorded as the best current.

2.3.4 Cutting planes and branch-and-cut algorithm

Now, we introduce the concept of *Cutting Planes*. Let P be the following linear programming problem:

$$max\{cx|x \in X\}$$

with

$$X = \{x \in R^n | Ax \leq b \ , \ x \in \mathbb{Z}\}$$

where A and b have rational coefficients. An inequality $\pi x \leq \pi_0$ is a *valid inequality* for X if $\pi x \leq \pi_0$ is valid for all $x \in X$. As a valid inequality for X, we have, in particular, those which define the *convex*

envelope of X, $conv(X)$, which is always a polyhedron. According to [38] the problem P could be solved through $max\{cx|x \in conv(X)\}$ because all the basic solutions that optimize this problem are also optimal solutions for P. However, there is usually no a simple way to characterize the inequalities needed to describe P .

Let S be the linear relaxation of P. We define a *cutting plane* as a valid inequality for X that cut the non-integer solutions of S. In general, an algorithm of cutting planes has the following form:

— *beginning* - $t = 0$ et $S^0 = S$

— *Iteration* t - Determine the optimal solution x^t for the relaxed problem: $max\{cx|x \in S^t\}$. If x^t is integer, then x^t is an optimal solution for P. Otherwise, if we find a valid inequality for X, $\pi^t x \leq \pi_0^t$, which cuts x^t, such as $\pi^t x > \pi_0^t$, we add it to S, $S^{t+1} = S^t \cap \{x : \pi^t x \leq \pi_0^t\}$, and we increase t, $t = t + 1$. If we do not find any valid inequality, we finish the procedure.

The *Iterations* are also known as the *separation problem* and give us the following theorem [39]:

Theorem 2.3.1. *cutting plane-based method on a $Ax \leq b$ system of constraints is polynomial if and only if the separation algorithm associated with $Ax \leq b$ is polynomial.*

A cutting plan procedure may intend to end when an entire solution is found, or serve only to improve the formulation of the original problem. We call *branch-and-cut* algorithm when the separation routine is within a branch-and-bound framework.

2.3.5 Column generation and the branch-and-price algorithm

In this section we discuss the *column generation* method. It is a technique used to solve a linear program when the number of variables of the model to solve is very large.

The basic principles of column generation have been published for more than four decades [40]. Hence, the methods of separation and the generation of columns (*branch-and-price*) were born. One of the first successful practical applications was made in 1961 and 1963 by Gilmore and Gomory on the cutting problem[41]. In this application, the solution was found by rounding.

In the generation method of columns, the original problem is divided into two problems:

— Restricted master problem: the problem to solve, with fewer columns;

— Subproblem: generator of new columns for the master problem.

These columns are embedded in the set of columns used by the restricted master problem to improve

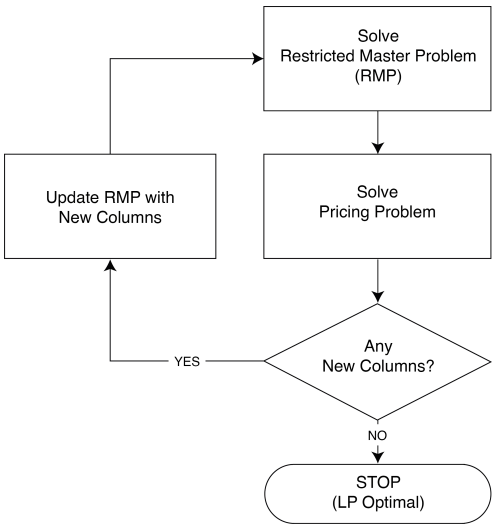


Figure 2.2 – Interaction between the restricted master problem and the sub-problem

the current solution. According to [40], the restricted master problem is a linear programming problem and does not have the integral property of the solution. Therefore, it still does not have an associated dual problem whose solution has exactly the same value, preventing the correct calculation of the costs of the dual problem. According to the same author, to obtain the dual values necessary in the solution of the sub-problem, the restricted master problem is not solved directly, so we must solve its linear relaxation. In this case, the variables corresponding to each column are no longer binary, but they remain as non-negative variables.

As pointed-out by [42], the purpose of the sub-problem is therefore to generate new columns (variables). These new columns are added to the restricted master problem that improves the value of the solution, or updates the dual variables, to produce other new columns by the subproblem. The interaction between these two problems is represented in the figure 2.2 below:

Since the main problem only works with a subset of columns, a linear programming packet can quickly find the solution. When new columns are added little by little, a solution can be obtained from the previous solution. To this point, the method focuses on approaches to solve the sub-problem more effectively. Let the problem P_3

$$min\ c^t x \tag{2.4}$$

subject to

$$Ax \geq b \tag{2.5}$$

$$x \geq 0 \tag{2.6}$$

This problem has a large number of variables. For this reason, it may not be possible to solve it directly. We first solve a problem P_4 which is smaller, with a subset $x' \subseteq x$ of variables:

$$min\ c'^t x' \tag{2.7}$$

subject to

$$A'x' \geq b \tag{2.8}$$

$$x' \geq 0 \tag{2.9}$$

For many constraints, each solution found must be tested if it satisfies all constraints, even those that are not yet in the model. We call this test as *viability test*. In the case where the problem has a large number of variables we must do the *test of optimality*, that is to say that each solution found must be tested to know if the solution is an optimal solution of the problem, even without variables that are not yet included in the model.

The optimality test can be performed by problem of separation of the corresponding dual PD_1 problem:

$$max\ by \tag{2.10}$$

knowing that

$$A^t y \geq c \tag{2.11}$$

$$y \geq 0 \tag{2.12}$$

When we solve the model P_4 we find a solution x'^*. Let y^* be the solution of the dual problem PD_1. This solution is tested to find the constraints violated by y^* in the dual set in (2.11). All or some of the variables corresponding to the violated constraints are added to the problem P_4, modifying x', c', and A'. This process must be iteratively redone until there are no more violated constraints in the dual problem PD_1 .

The test in (2.11) for a solution y^*, rewritten as $c - A^t y \geq 0$, shows more clearly the significance of the separation problem in the dual model. For each variable x_j, the first term of the inequality (2.13) is the reduced cost in the primal problem P_4, computed by the dual prices y_i^* of the constraints.

$$c - \sum_i a_{ij} y_i \geq 0 \tag{2.13}$$

It should be noted that an unsatisfied constraint in the dual problem is equivalent to a negative cost variable negative in the primal problem. This variable, when added to the model, can produce a solution having a better value in the objective function, and it is therefore interesting to add it to the problem. This method is called *column generation* and we call *branch-and-price* algorithm when the column generation routine is within a branch-and-bound framework.

The original problem with fewer variables is called *master problem*. The separation problem in the dual (or *pricing*) model, which is the problem of finding or generating a new column, is commonly called *sub-problem*. As mentioned earlier, this process can be used in problems where the number of variables is very large, which would make it impossible to develop a model with each of them exactly. According to [43], it is not necessary to introduce all the variables in the model, just create an algorithm that, starting from the dual prices of the current solution, finds variables of reduced cost, then simply add these variables to the master problem. We can go even further: it can be applied to problems where we do not yet know all the variables. It is enough to generate iteratively the variables of reduced negative cost which are interesting in the problem of optimization.

2.3.6 Heuristic methods

A heuristic is an algorithmic procedure developed by a cognitive model, usually by rules based on the experience of developers. Unlike exact methods that seek to find an algorithmic way to find an optimal solution by combining or looking for all possible solutions, heuristics usually tend to have some degree of knowledge about problem behavior, generating a much smaller number of solutions [44].

Many factors make using heuristic algorithms interesting to solve a particular problem. According to [45], some factors are:

— When there is no exact method to solve this problem or it requires a very high processing time. In this case, offering a good solution is better than having no solution;

— When the solution is not necessary because the solutions obtained are already reasonable;
— When data is unreliable. In this case, the search for the optimal solution has no meaning, because it is already an approximation of reality;
— When time and/or money constraints require the use of rapid response methods;
— As intermediate steps of other algorithms, potentially accurate or others heuristics.

Heuristic methods include strategies and approximation procedures to find a good solution, even if not optimal, within a reasonable calculation time.

A *constructive heuristic* is a type of heuristic that begins with an empty solution and extends the current solution until a complete solution is obtained. It differs from the local search heuristic which begins with a complete solution and then tries to improve the current solution with local movements [45].

A *meta-heuristic* is a generic heuristic method for solving optimization problems (usually in the domain of combinatorial optimization). Meta-heuristics are generally applied to problems that do not have a known effective algorithm. According to [46], they use the combination of random choices and historical knowledge of the previous results obtained by the method to orient itself and carry out its search through the space of possible solutions, which prevents premature stoppages in optimal premises.*Local search* is a meta-heuristic method for solving optimization problems. The main objective is to find the ”best” solution among several possible candidates by maximizing or minimizing the criteria. From a starting point, this type of method applies local changes until a solution considered ideal is found, or a time limit of execution is exceeded [46]. Therefore, each solution found, better than the last, gets the new starting point for the next iteration. The search space contains all possible solutions to a problem.

According to [47], the criteria for the use of a heuristic to solve a particular problem are the following:

— **Optimality**: When there are several solutions to a given problem, does the heuristic ensure that the best solution will be found? Is it really necessary to find the best solution?
— **Completeness**: When there are several solutions to a given problem, can the heuristic find all of them? Do we really need all the solutions?
— **Precision**: Can this heuristic provide a good interval of confidence for the desired solution? Is the gap between the optimal solution and the one found by the heuristic good enough?

— **Runtime**: Is this the best-known and fastest heuristic to solve the problem? Some heuristics converge faster than others. Some heuristics are only slightly faster than conventional methods.

In some cases, it may be difficult to decide whether the solution found by the heuristic is good enough because the underlying theory of this heuristic is not very elaborate.

Chapter 3

A view on the telecommunications network evolution

Contenu

Every second, a large amount of digital data is transported through huge number of types of devices via cellular networks worldwide, and expectations are at a greatly accelerated growth with increasingly large requests. In this context, in few years these networks could reach their maximum capacities in terms of data transmission. Consumer mobile communications, video downloads, and the use of mobile applications represent the bulk of the current use of radio resources in 4G networks. Hence, the need to optimize resources in classical cellular networks becomes extremely important. To

face all these challenges, 5G technology is posed by enabling the digitization of society and economic information. The idea behind the concept of 5G is that it does not correspond to a simple increase in data rate, as was the case for previous generations. With 5G, the aim is thereby to widen the diversity of users.

In this context, Device-to-Device communication (D2D) has been highlighted, as it provides a better allocation of resources compared to conventional cellular communication underway. D2D is a set of communication technologies that allow two digital devices (i.e., mobile phones and robots) in close proximity zone to communicate directly with each other without passing through the traditional cellular network's access points (i.e., antennas and base stations). With discovery and communication functions, the user equipment can find others around to exchange information.

Network Slicing [4] is another novel virtualized infrastructure model in new generation mobile networks. This technology now not only covers application-level slice abstraction as done with preliminary works on 'slicing', but also physical and switching layers virtualization, with different radio access and link communication technologies. This transition challenges slice network design since multiple resources and segments, historically managed independently from each other, are to be operated with continuity in networking and computing resource allocation and provisioning as a whole and unique service. In this context, different providers can be associated with different communication services running on the same physical network at the access, core, and application segments.

Because of different bitrate and latency requirements, policies on radio access function splitting have an impact on the backhauling network dimensioning, and therefore on the placement of core network functions and on the configuration of edge computing application servers. Moreover, different policies for control versus data-plane function sharing and scaling are to be applied. For instance, in 5G a first service classification in three classes is given [6]: enhanced Mobile Broadband (eMBB), Ultra-Reliable Low Latency Communications (URLLC), and massive Machine Type Communications (mMTC). Each of these application categories has its specific requirements, such as maximum latency, minimum availability, and bandwidth capacity and, to provide a flexible environment to support those customized networks, novel infrastructures are supported by Network Function Virtualization [2], Software Defined Networking [3], and Network Slicing [4] technologies. Hence, each communication service (CS) provider can deploy its services on top of logical networks, named Network Slices, specifically tailored to its technical requirements.

Addressing end-to-end network slicing, however, requires considering heterogeneous resources from different physical and virtual network topologies, each with specific technical constraints and particular orchestration rules. Furthermore, an important novelty of 5G specification is the introduction of three novel mapping dimensions influencing the placement and interconnection of slices and network functions: (i) a CS can be delivered by multiple network slices; (ii) Slices can be decomposed into Network Slice Subnets; and (iii) Network Functions can be decomposed into Network Function Services. While the first mapping requirement can simply impact network design hyperparameters only, the second and third ones come with new technical constraints to guarantee a coherent provisioning of each CS. Namely, continuity constraints among slice subnets and the capacity to support specific behaviors for all the components of the same slice, such as function splitting, sharing, and scaling policies. In addition to these peculiar constraints, classical network function embedding, routing, and requirements on latency, availability, and network and computing capacities hold as well.

In this chapter, we present the entities appearing in new generation networks and the main modeling aspects and technical constraints related to 5G systems and beyond[1].

3.1 Core network evolution

Third Generation Partnership Project (3GPP) emerged from the need of technically specifying the third generation (3G) mobile network. Such specifications were structured in documents called Releases: the first one was issued in 1999, called Release 99 [48]. At that time, 3GPP's work was based on Global System for Mobile communication networks, also called 2G systems. The Core Network is responsible for all control and central processing for the 3G system, being an evolution of the legacy Network Switching Subsystem. For the conventional Circuit Switched domain, the 3G core network was conceived with Mobile Switching Center (MSC) and MSC Gateway (GMSC) entities, responsible for carrying calls from network users and interfacing with external circuit-based networks, respectively.

3G mobile technology supports a wide variety of services, ranging from multimedia applications available on the Internet. For this purpose, 3G core is also conceived with a Packet Switched domain

1. The content of this chapter was partially published in the following papers: W. d. S. Coelho, A. Benhamiche, N. Perrot and S. Secci, "Network Function Mapping: From 3G Entities to 5G Service-Based Functions Decomposition," in IEEE Communications Standards Magazine, vol. 4, no. 3, pp. 46-52, 2020.
W. d. S. Coelho, A. Benhamiche, N. Perrot and S. Secci. "On the impact of novel function mappings, sharing policies, and split settings in network slice design." 2020 16th International Conference on Network and Service Management (CNSM). IEEE, 2020.

to carry users' data. As described by [49], this domain includes General Packet Radio Services Support Node (SGSN) and Gateway General Packet Radio Services Support Node (GGSN). While SGSN is the entity responsible for mobility, session management, and billing, GGSN can be seen as the main entity of the 3G core. In this architecture, GGSN is responsible for ensuring and managing the connection with external packet-switched networks, (e.g., Internet). Furthermore, Equipment Identity Register (EIR), Home Location Register (HLR), and Authentication Center (AuC) are the entities shared by both Circuit and Packet domains and contain all administrative information of each subscribed User Equipment (UE). These entities are also responsible for connection rules and information and data protection.

3.1.1 From 3G to 4G

To better deal with the immense popularization of the Internet on mobile phones and to support a greater number of customers and volume of data, Release 8 [50] was delivered by 3GPP in 2009. Release 8 was the first document presenting standards for Long-Term Evolution (LTE) mobile system, so-called 4G. It is important to note that 3GPP did not change its name when moving to 4G (and beyond) technology specification. .

Release 8 presents standards for the Packet Switch system, called Evolved Packet Core (EPC). Within this new architecture, all services (e.g. voice, SMS, and data) are driven by the Internet Protocol (IP), which implies the disappearance of Circuit Switched domain from legacy Core Network. As depicted in Fig. 3.1, MSC and EIR's functionalities were combined into Mobility Management Entity (MME), the main controlling entity in EPC. MME is responsible for authentication, roaming, and session management on the broadband network. It interfaces with two other entities, named Home Subscriber Server (HSS) and Serving Gateway (S-GW). The former is the database where all network user information is found and can be considered as an aggregation and evolution of HLR and AuC from the 3G system. S-GW, in turn, has some functionalities from the legacy SGSN 3G core entity. It acts as mobility anchor during handovers and is responsible for routing users' data packets. In that way, each user that accesses the EPC is associated with a single SGW, which interfaces with Packet Data Network Gateway (PDN-GW). This last entity is the essential interconnection point between EPC and external IP-based networks. It is also responsible for packet filtering, legal interception, and IP address allocation. Hence, PDN-GW has functionalities from GMSC and GGSN, combined.

Finally, we find the Policy and Charging Rules Function (PCRF) within the EPC system. It is part of a functional framework called Policy and Charging Control (PCC), which is used to control the bearer services offered by the LTE network as well as to control the charging mechanisms. PCRF is a software component and is responsible for providing access rules and policies for core network gateways.

3.1.2 Separating 4G core with CUPS

In order to offer a higher degree of modularity and scalability for core entities, 3GPP proposes a new architecture called CUPS: it stands for Control and User Plane Separation and offers a complete split between Control Plane (CP) and User Plane (UP). In this new EPC system, introduced in Release 14 [6], a more effective approach to support increasing and dynamic traffic is provided. With CUPS, UP and CP scaling can be made independently. For this purpose, S-GW becomes Serving Gateway Control Plane function (SGW-C) and Serving Gateway User Plane function (SGW-U). Equally, PDN-GW is split into PDN Gateway Control plane function (PGW-C) and PDN Gateway User Plane function (PGW-U). While SGW-C and PGW-C are responsible for controlling the processing of data that transverse User Plane functions, SGW-U and PGW-U implement specific features for a certain group of users, such as low latency or high bandwidth, to ensure Quality of Service (QoS). 3GPP has also proposed to split Traffic Detection Function (TDF) into CP and UP entities. TDF is part of PCC block [51] and is responsible for improving the operator's charging functions. With CUPS, TDF becomes TDF User Plane (TDF-U) and TDF Control Plane (TDF-C) functions.

3.1.3 From 4G to 5G

The 5G architecture enhanced by Network Function Virtualization (NFV), Software-Defined Networking (SDN), and Network Slicing provides an even more flexible environment supporting end-to-end customizable virtual networks. With dedicated resources to ensure specific QoS for a wide variety of services, all entities in 5G might be entirely or partially virtualized. 3GPP proposes 5G Core (5GC), another change in the mobile core network. Hence, 5GC presents the evolution of the legacy entities found in 4G and additionally introduces optional Network Functions that might be necessary to control and operate each slice. In the resulting new virtualized architecture, what were called entities in 3G and 4G are now called Network Functions in 5G

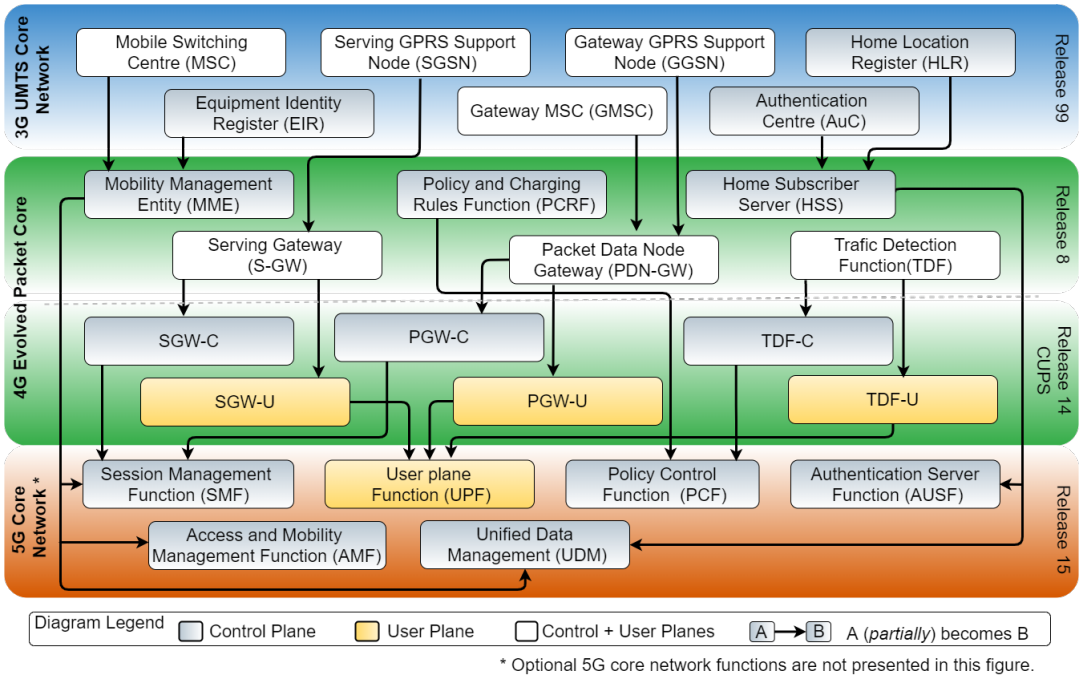


Figure 3.1 – 3GPP Network Core function evolution: from 3G to 5G.

As depicted in Fig. 3.1, MME functionalities are split into three new functions: Access and Mobility Management Function (AMF), Session Management Function (SMF) and Unified Data Management (UDM). AMF is responsible for registration, connection and reachability management, access authentication, and access authorization. UDM further shares responsibilities found in legacy HSS with the new Authentication Server Function (AUSF). SMF, in turn, is responsible for session management and takes all functionalities from PGW-C and SGW-C. Additionally, SMF will directly interface with 5G User Plane, which is represented by User Plane Function (UPF). In this context, UPF might apply all functionalities found in PGW-U, SGW-U and TDF-U, combined. Moreover, policies and charging rules, which were in 4G PCRF entity, are provided by the Policy Control Function (PCF), which interfaces with Application Function (AF); AF is meant to act like the logical 4G AF integrated into PCC framework [51], offering dynamic traffic information and applications based on user behaviors.

In this evolving context, we can identify a primary set of functions that need to be present to deploy a core slice. In that way, this slice must contain at least the basic functions connected to control and user planes: AMF, SMF, PCF, AUSF, UDM and UPF. Additionally, to better control and operate each slice in future 5G networks with specific requirements and technical constraints, several new functions are proposed by 3GPP [8, 52, 53]. Following, we present the main functionalities of these

new functions:

- *Network Exposure Function (NEF)* - event and capability safe exposure to functions and third party networks.
- *Non-3GPP Inter Working Function (N3IWF)* - IP Security tunnel support in cases of untrusted non-3GPP accesses.
- *Network Slice Selection Function (NSSF)* - slice selection for serving users.
- *Unstructured Data Storage Function (UDSF)* - information retrieval and storage as unstructured data.
- *5G-EIR* - Permanent Equipment Identifier status checking.
- *Network Data Analytics Function (NWDAF)* - slice load level information.
- *Charging Function (CHF)* - online and offline charging
- *Network Repository Function (NRF)* - provides NF instances information such as function type, function ID, and Network Slice related Identifier(s).
- *Unified Data Repository (UDR)* - subscription and policy structured data retrieval and storage.
- *SMS Function (SMSF)* - SMS management and delivery over Non-Access-Stratum.
- *Location Management Function (LMF)* - measures user, downlink and uplink location.
- *Security Edge Protection Proxy (SEPP)* - non-transparent proxy responsible for filtering and policing between Public Land Mobile Networks' control planes.
- *Binding Support Function (BSF)* - supports binding information updates by NF service consumers.

Figure 3.1 summarizes the aforementioned core network evolution, from 3G to 5G. For an in-depth functionality description of each 5G function, one may refer to [8, 52, 53].

3.2 5G core specificities

To better support network slicing, 3GPP has proposed more flexible, customized service-based network functions for 5G systems. With this new architecture, functions make use of a complete integrated interface that allows them to expose their capability to other authorized NFs without precluding the use of intermediate functions. This implies that all Control Plane functions have potential to directly communicate with each other by request/response and subscribe/notify application-level signaling. For this propose, each CP function is conceived with a set of NF services to be consumed

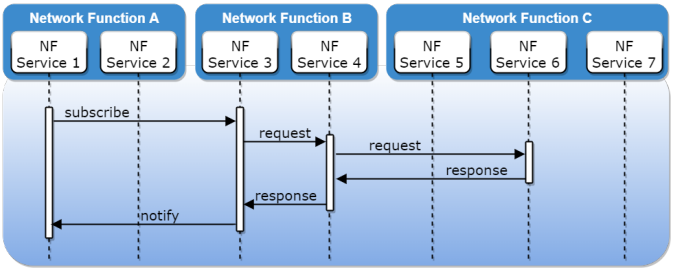


Figure 3.2 – Service-based interactions between network functions.

by other functions. The interactions between CP functions are depicted in Fig. 3.2. In this example, A is a *NF service consumer* from B's point of view, which is, in turn, a *NF service producer* from A's perspective. Note that the same function may be service consumer and service producer at the same time, as illustrated by B. One function might consume services from different NF service producers and a function might also offer services for different NF service consumers. It is also important to notice that a NF service can provide and consume services to and from other NF services within the same function.

3GPP has also been providing some technical reports with a non exhaustive list of possible interactions between different functions and NF services [8, 54, 55]. Some of those iterations are exemplified on Table 3.1, where UDM is taken as example with its non-exhaustive list of NF services [55]. Note that UDM is able to offer a same service to different NF service consumers. In this example, UDM can subscribe data from AMF, SMF and SMSF. It is also important to notice that a same service might make different operations (e.g. get, subscribe, update, result confirmation) and they are made by request/response and subscribe/notify queries.

Table 3.1 – Example of a NF service decomposition: UDM as NF service producer

NF Service	NF Service operations	Signaling semantic	NF Service consumers
Subscriber Data Management	Get	Request/Response	AMF, SMF, SMSF
	Subscribe	Subscribe/Notify	
	Notification		
UE Context Management	Get	Request/Response	NEF, SMSF
	Update		
UE Authentication	Result Confirmation		AUSF
Parameter Provision	Update	Request/Response	NEF

3.2.1 Customized service-based network functions

Legacy mobile networks lack the ability to provide a specific set of functionalities for each service proposed by operators, each one with a very specific technical requirement. 5G systems, leveraging NFV and SDN to support flexible data and service connectivity with an efficient deployment of customized slices, can offer a minimal and adapted set of core NFs for each case: each function can be programmed to contain only the NF services required to keep each slice operational and efficient. Hence, two UPFs in different slices, for example, might contain different capabilities as well as different customized sets of NF services. Having minimal sets of adapted functions implies a drastic reduction on the use of limited physical resources.

To assure all these features, each NF service must be reusable, self-contained, and use individual schemes independently from all NF services present in the system. Effectively, being reusable eliminates redundancy by offering total integration between all others NF services, within the same NF or not. In that way, each service is specific to a NF service producer and must be available to all potential NF service consumers. Additionally, all NF services are supposed to be self-contained and their operations must be done in their own contexts. Hence, NF services are allowed to operate only on their own internal storage, using their own software resources.

3.2.2 Auto-scaling and distribution

To better deal with traffic load fluctuation, auto-scaling becomes a powerful tool to alleviate network congestion due to the high data traffic load in certain periods. Additionally, distributing the flow intelligently is essential to ensure that each user has access to the demanded Communication Service with assured QoS. Such a distribution must be done by frameworks capable of detecting data traffic load fluctuation in a given slice in order to multiply instances of some critical functions. By knowing the high signaling sent and received by a sub-set of NFs within a slice, the orchestrator entity is responsible to provide and implement a traffic-aware algorithm capable of creating NF instances in critical moments (scale in) and distributing the traffic among them. Equally, in periods of low data traffic load, NFs copies in the same slice are removed (scale out) and a data traffic redistribution must be done. Following the 3GPP standards, physical resources can be saved and QoS is assured to users when NF instances are scaled out and in, respectively, once data traffic is properly distributed.

In 5G, scalability is facilitated by using as much stateless functions as possible. Today's applications found in mobile networks are often stateful, which greatly increases the risk of failure and related management overhead. For the 5G core, most of its functions are meant to be stateless, facilitating scalability and decreasing the failure risk. To do this, these functions use other specific NFs that store states (e.g., network and users' data and information). For such management, 3GPP proposes UDM, UDSF, and UDR functions.

3.2.3 The user-plane case

Even though there is no precise reference to the User Plane domain, some assumptions based on 3GPP releases can be taken into account when designing a slice that offers UPFs. Firstly, one possibility is to install a single instance of UPF with all the necessary NF services to a specific slice. This approach becomes necessary in slices dedicated to civil security; in this example, security and isolation constraints might imply that there must be only one node between access and data networks. However, for those scenarios where there are no such constraints, proposed 5G systems support interconnections between different UPFs by reference point named N9 [8]. In this way, each UPF would be conceived with a sub-set of NF services and, unlike control plan data flow, user data is carried through an ordered chain of functions, each one responsible for a specific data processing. In the case of downlink data buffering, for example, one UPF may act as the PDU Session Anchor while another UPF may be responsible for data encapsulation. Furthermore, N9 reference points can be established between different UPFs within the same Public Land Mobile Network (PLMN) or different PLMNs [8] and, consequently, between different slices.

3.2.4 Functional splitting in the radio access network

Flexible Radio Access Network splitting [56] is a technique meant to increase network efficiency leveraging NFV flexibility. In 1G and 2G RANs, all entities responsible for radio and baseband processing were distributed and integrated into each base station. To minimize costs and facilitate network deployment, it was proposed to split the base station into Remote Radio Unit (RRU, also called Remote Radio Head and Radio Unit) and Baseband Unit (BBU) (also called Data Unit). In this context, the RRU is responsible for Physical Layer functionalities, while the BBU is responsible for Data Link Layer functionalities [57]; the distance between these two entities could be up to 40

kilometers. However, there is still redundancy in the network: all RAN functionalities are replicated for each pair of BBU and RRU. To overcome this, centralized RAN (C-RAN) was first introduced in 2011 [58]; pools of BBUs with large capacity, now called Centralized Units (CUs), are proposed to treat traffic from a sub-set of RRUs, now named Distributed Units (DUs). Hence, one of the first tasks is to define the functionalities enabled locally at the DU, and those installed centrally at the CU and thus shared among a subset of DUs.

Figure 3.3 illustrates different functional split options on the 4G stack, as the 5G RAN split options have not yet been specified. Let's take option 3 as an example: all functions from Radio Frequency (RF) to Low Radio Link Control (RLC) blocks are locally installed, while high RLC, Packet Data Convergence Protocol (PDCP) and Radio Resource Control (RRC) functions are centrally installed. Equivalently, with option 7 on the uplink direction, all functionalities after the low Physical (PHY) block are installed at a CU, while with option 5 all entities before the low Media Access Control (MAC) block are installed at the DUs.

Since the functional split was originally meant to be made *a priori* (i.e., before deploying the network) choosing the best split [59] for each scenario is not trivial. Defining the distributed and centralized functionalities should take into account end-to-end delay and total bandwidth constraints on each physical path connecting DUs and CUs while optimizing the resource allocation. It is important to mention that all distributed functionalities should be installed in each DU to support any type of split. Complementary, centralized functionalities have few instances that are installed in CUs and are shared by a specific sub-set of DUs. The dependency factors such as varying network latency and capacity has recently motivated experimenting *dynamic* functional splitting, where the split decision can be reconfigured on a short time scale for one or a few split options [60].

Table 3.2 depicts different fronthaul (FH) bitrates and latency indicators for each functional split. The bitrates are calculated as in [59] for a scenario using 100 MHz bandwidth and 32 antenna ports, while the maximum accepted one-way latency through FH is proposed by 3GPP [57]. Note first that the highest bitrates and lowest latency are imposed by option 8. However, one of the advantages

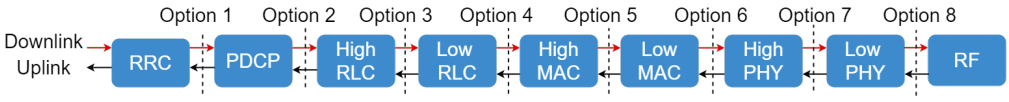


Figure 3.3 – Different functional split options.

Table 3.2 – Fronthaul bitrate and latency in functional split.

Functional Split	DL Bitrate	UP Bitrate	FH Latency
Option 1: RRC-PDCP	4 Gbps	3 Gbps	10 ms
Option 2: PDCP-hRLC	4 Gbps	3 Gbps	1.5-10 ms
Option 3: hRLC-lRLC	4 Gbps	3 Gbps	1.5-10 ms
Option 4: lRLC-hMAC	5.2 Gbps	4.5 Gbps	0.1-1.0 ms
Option 5: hMAC-lMAC	5.6 Gbps	7.1 Gbps	0.1-1.0 ms
Option 6: lMAC-hPHY	5.6 Gbps	7.1 Gbps	0.25 ms
Option 7: hPHY-lPHY	9.2 Gbps	60.4 Gbps	0.25 ms
Option 8: lPHY-RF	157.3 Gbps	157.3 Gbps	0.25 ms

of choosing this split would be in reducing the number of NFs throughout the access network, as they would be installed centrally and shared by different DUs. Contrarily, option 1 requests low bitrates and admits higher latency; the disadvantage of this option is that almost all NFs would be installed locally - this scenario demands higher computational power on each DU, which could be impractical given the number of expected DUs in mobile systems. It is also important to point out the difference between downlink (DL) and uplink (UP) bitrates, due to physical layer operations (e.g., transformations between transport blocks and in-phase and quadrature symbols in each direction of the data flow).

Figure 3.4 represents a scenario with different split options with two operators and a RAN function chain composed of four NFs. In this example, the split between NF 1 and NF 2 is applied to treat the flow from DU 3 and DU 4. These two DUs have only NF 1 installed locally and send their flow to CU 2, which has NF 2, NF 3, and NF 4. Note that these functions in CU 2 are shared by both DU 3 and DU 4.

SDN and NFV technologies can be used together with C-RAN to offer flexibility to split RAN slice subnets [6, 57]. To this propose, two classes of RAN functions are proposed by [61]: *asynchronous*

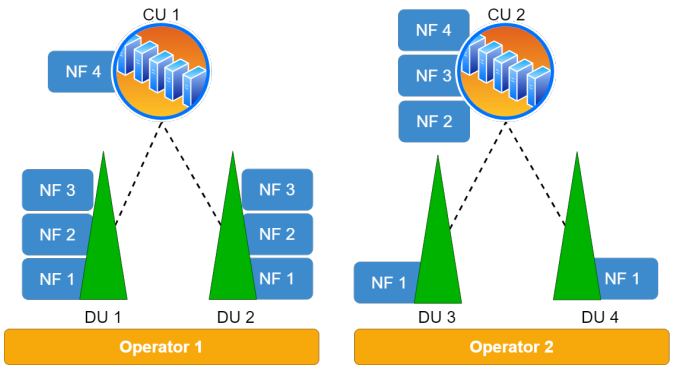


Figure 3.4 – Functional split example with four RAN NFs.

network functions and *synchronous network functions*; the former refers to network functions that process data asynchronously with the radio interface and demand low data rates. State transition and handover preparation are functionalities from RRC and PDCP blocks that are candidates to be virtualized, centralized into CUs pools, and shared by a sub-set of DUs. However, time-synchronous functions, such as interference coordination, scheduling, and power control from PHY and MAC blocks, process data synchronously with the radio interface, requiring low latency and high data rate. Hence, the related NFs might need some hardware acceleration, which implies that they are good candidates to either be implemented as dedicated machines or installed on a path that assures low latency and high bandwidth. According to [62], strict timing dependency between protocol layers must be avoided, using instead asynchronous NFs as much as possible to grant more flexibility to RAN slicing.

Being consistent with [6, 57, 62], we incorporate flexible RAN splitting in order to design end-to-end network slices. This approach can better deal with the heterogeneous requirements of each NS request while decreasing the redundancy in the network, that is, minimizing the number of virtual AN-based functions installed throughout the physical network. [63, 64, 56] address the challenges of flexible functional split schemes to optimize the allocation of physical and virtual resources. [56], for example, proposes a new architecture that introduces a flexible split of RAN functionalities between the Cloud-RRH, an edge cloud, and the central cloud. [64], in turn, analyze the technical features of the network in order to find the optimal split for different scenarios; the authors considered the configuration of the base stations, the fiber ownership, and the data transmission direction. They demonstrated that a lower total cost of ownership can be achieved with optimal functional split compared to classical radio access networks. The required backhaul capacities for uplink traffic in terms of minimum bandwidth and maximum latency for different split options are analyzed in [63].

Even though there are several works in the literature addressing functional split mode selection and network slicing problems with NF sharing and NF scaling, no attention has been given to jointly address the complete problem in order to design end-to-end network slices including functional splitting for the radio access functions and different schemes for dimensioning and sharing NFs.

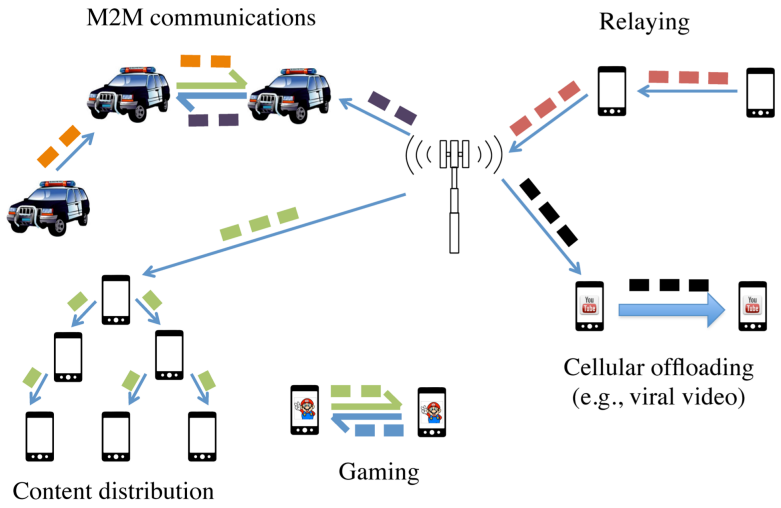


Figure 3.5 – Representative use-cases of D2D communications in cellular networks.
Source: [65]

3.3 Device-to-device communication

In new generation access networks, mobile User Equipment will be able to host functions that give them new abilities such as sharing connectivity, capacity, and CPU resources with other UEs, regardless of the ongoing traditional communications. The 5G wireless technology, along with the evolution of mobile users' behavior and needs, will make the current scheme of communication (UE to Base Station) no longer optimal in terms of radio resource utilization for some cases. The Device-to-Device (D2D) communication mode is one of the new approaches presented as a promising alternative to traditional communication in cellular networks. D2D communication is defined as direct communication between two mobile or fixed user devices, without traversing the Base Station (BS) [65]. This technology allows to reuse radio resources and to decrease the end-to-end latency of local communications. Then, D2D would allow a set of UEs geographically close to each other to establish direct D2D communications, or span multiple links (multi-hop D2D communications), to access a given service (e.g. video streaming or gaming) while ensuring the required service quality. Figure 3.5 illustrates the advantages and uses of D2D communication.

D2D communication can be divided into two groups: *Inband D2D* and *Outband D2D*. The former proposes the use of licensed spectrum [66]. It is further subdivided into *Underlay D2D Inband*, where both D2D communication and classical cellular communication can share all licensed spectrum range,

and *Overlay Inband D2D*, in which the spectrum range is divided into two and each part is dedicated to only one type of communication. On other hand, Outband D2D proposes to use unlicensed spectrum [67]. It is also further subdivided into two others sub-categories: *Controlled Outband D2D*, in which coordination between radio interfaces is controlled by the Base Station, and *Autonomous Outband D2D*, where coordination between radio interfaces is controlled by the users.

One of the first works proposing the use of D2D communication was published in 2000 [68]. Soon afterward, several researchers turned their attention to this promising technology and looked for effective methods to improve its use. Both Inband D2D and Outband D2D are studied since then, with more attention to the former. However, several authors suggest that the main advantages of Outband D2D are related to the absence of interference between cellular and D2D users. In addition, there is no need to dedicate cellular resources to the D2D spectrum as in the Overlay Inband D2D approach. Hence, resource allocation becomes easier because the scheduler does not require to take the frequency-time matrix and location of the users into account. In addition, simultaneous D2D and cellular communication are feasible. However, according to [65, 69] main disadvantages related to Outband D2D commutation are:

— Interference in unlicensed spectrum is not in the control of the BS;

— Only cellular devices with two radio interfaces (e.g., LTE and WiFi) can use Outband D2D communications;

— The efficient power management between two wireless interfaces is crucial, otherwise, the power consumption of the device can increase; and

— Packets (at least the headers) need to be decoded and encoded because the protocols employed by different radio interfaces are not the same.

On the other hand, according to the authors in [65, 69], the main advantages and problems related to Underlay Inband D2D are:

— Spectrum efficiency: With the use of D2D we can improve the use of the cellular spectrum very effectively. This can be done by using interference reduction techniques between D2D and classical cellular communication, or by developing high interference recognition applications to avoid the use of problematic frequencies in a certain region and time.

— Power efficiency: In several studies, it was observed that power efficiency can be improved with D2D technology. One of the most used techniques is the dynamic choice between cellular or

D2D as a communication mode.

— Performance with QoS/Power Constraints: Improving the performance of D2D-enabled cellular systems with QoS/power constraints usually require advanced techniques such as stochastic optimization, nonlinear programming, and integer optimization.

However, Overlayng Inband D2D eliminates the concern about the interference between the two types of communication, since they do not share the same spectral range. This approach considerably reduces the number of resources to promote each part, and may promote an important waste of them. BS-assisted scheduling [70] and D2D power control [71], techniques were proposed to reduce D2D interference. Another technique is proposed by the authors in [69]. They propose to use the Base Station as a relay (backup re-transmitter) for the D2D transmission or to use multiple D2D users as the relays (re-transmitters) for multi-casting. According to the authors, both methods have low complexity which makes them practical for real-world scenarios.

Even with important works using Controled Outbound D2D [72, 73, 74, 75, 76], Autonomous Outband D2D [77, 78], and Overlayng Inband D2D [70, 71, 69], most of works were developed using Underlay Inband D2D.

In [79], Underlay Inband D2D was applied in both non-orthogonal and orthogonal resource sharing modes. In this work, the authors showed how to find a sub-set of viable solutions to ensure the optimal solution for the non-orthogonal resource sharing technique. They have shown that their approach provides gain over pathloss-based selection mode. Due to the non-convex difficulty found in optimization problems in this context, authors in [80] proposed two sub-optimal strategies that have a smaller complexity, facilitating their resolution. In such a work, authors used uplinks in Underlaying Inband D2D technique to optimize only the transmission power of the D2D communications, fixing in advance the transmission powers of the classic colleges that use base stations.

Authors in [81] propose dividing each cell into inner and outer regions and allocating different frequency bands for the communications regarding the position of the users. The authors present a fractional frequency reuse (FFR) approach to minimize interference between cellular communications and D2D. Following the approach depicted in Figure 3.6, authors in [81] showed that D2D can significantly improve the total throughput of the cellular network compared to random allocation schemes. Authors in [81] were able to improve the spatial-spectral efficiency in the network using the inter-cell interference coordination (ICIC) technique, which is managed by the base stations.

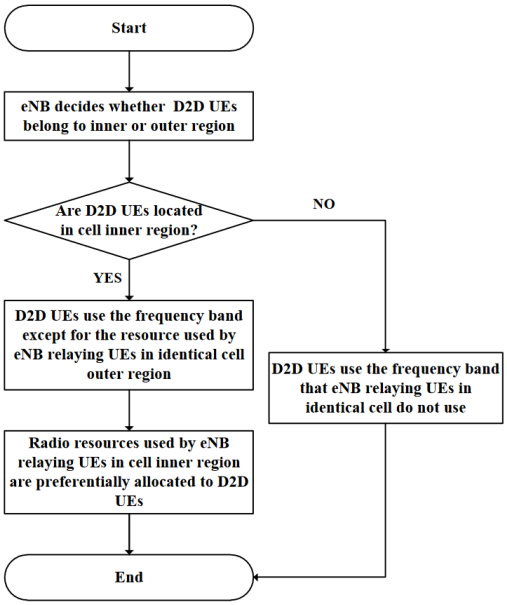


Figure 3.6 – Resource allocation prodedure for D2D UEs
Source: [81]

Finally, authors in [82] propose distance-constrained resource-sharing criteria as select mode technique, controlling the interference by keeping a minimum distance between communications that use the same frequency. Numerical results have shown the effectiveness of this approach to significantly reduce the interference caused by D2D active links.

3.4 5G system mapping requirements

The 3GPP specifications [8, 7, 6] present the different entities appearing in 5G systems; as we describe in [1], they are: User Equipment (UE), Communication Service (CS), Network Slice (NS), Network Slice Subnet (NSS), Network Function (NF), NF Service (NFS).

Fig. 3.7 depicts the relationships between these entities. A UE can be a smartphone, a robot, or even an autonomous car, that might be connected to several CSs; e.g., a car connected to an Autonomous Car Service while broadcasting movies and music from Streaming Services to its passengers. To better deal with heterogeneous technical constraints of each service, each CS might run on one or more customized NSs. Additionally, each NS might be composed of one or more NSSs, which might also be composed of lower-layer NSSs. A simple example of this scenario is given by considering an NS

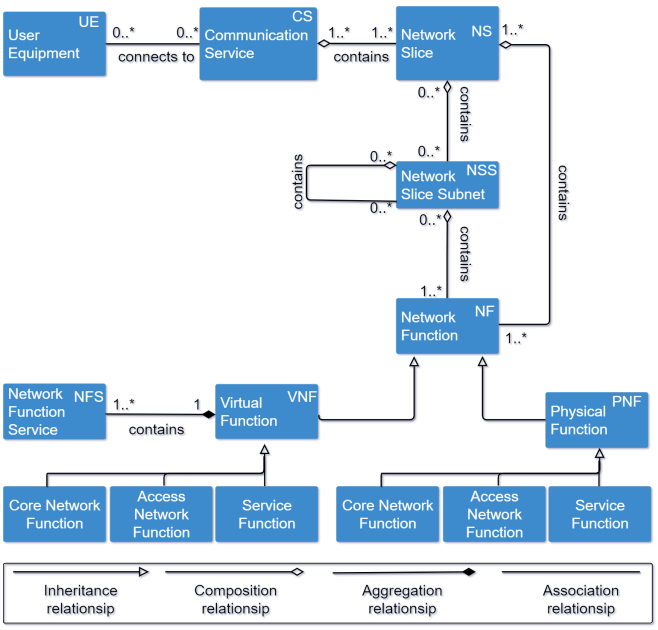


Figure 3.7 – Relationships between 5G entities. Source: [1].

composed of an access NSS and a core NSS, where the latter can, in turn, be composed of a control-plane NSS and a data-plane NSS (data-plane relates to user application traffic while control-plane traffic involves network and service signaling functions).

In this nested architecture, each NS or NSS is composed of one or more NFs attached to the Access Network (AN; e.g. Scheduler Function and Connection Mobile Control Function) or to the Core Network (CN; e.g. Session Management Function and Access and Mobility Management Function), or representing a Service Function (e.g. Firewall, Proxy, and Load Balancer). At the lowest level, each virtual NF is composed of a set of NFSs.

We can distinguish five mapping levels for creating a complete virtual 5G environment, each one with its characteristics and technical complexities to be implemented.

3.4.1 Mapping NF services into network functions

The first mapping is the most technical one. At this level, the difficulty lies in how to program each function so that it behaves exactly as expected. As already discussed, each function is conceived with a customized sub-set of NF services that keeps a slice operational and efficient. The difficulty related to this mapping is imposed by the need of minimizing the resource allocation for each function.

Intuitively, the greater is a sub-set of NF services within a function, the greater is the number of physical resources that must be provided to install it. Clearly, an important work must be done on this level to find the right sub-set of NF services for each function and each use case, assuring the operability and efficiency of each CS.

3.4.2 Mapping network functions into slices and slice subnets

Mapping functions into slices is another technical problem. In this mapping, each NF has its own minimal physical capacity request (e.g. CPU, RAM, storage) as well as its capability (e.g. package treatment rate). The mapping at this level must also take into account the connection between each possible pair of functions that must be linked, as well as the required capacity and processing rate on each of those virtual links. At this level, the orchestrator entity is already aware of the sub-set of NF services within each function. As a result, mapping functions into slices offers to the orchestrator entity a catalog of well-defined slice templates to specific and recurrent CSs. The problem for this mapping level is how to decide which function must be present in each slice and how to predict the quantity of each NF instance within a slice to better deal with the expected data rate throughout the system. It is important to mention that this level also refers to the mapping of functions into slice subnets, so creating NSSIs.

3.4.3 Mapping slices subnet into slices

After creating a slice subnet catalog as mentioned in the previous mapping level, the orchestrator entity might decide to create NSIs from well-defined slice subnet templates. This may be the case where a core NSI is created from UP and CP slices, combined. Hence, from 3GPP's point of view, those two last NSs are seen as slice subnet and the whole core as a slice. Notably, to do this type of mapping, well-defined sets of slice subnet templates must be available to guarantee the deployment of a whole NSI.

3.4.4 Mapping slices into communication services

This fourth level mapping is less technical than the previous ones. It takes advantage that all slice templates are already cataloged and ready to be deployed. According to its heterogeneous needs and expected data rate throughout the slice, each CS has a subset of possible slice templates to be mapped

to. In this context, one may use matching techniques to better identify which templates are the most appropriated to a given set of CSs. It is worth noting that this mapping level can also be applied when slices are already deployed. In this context, each CS request might be mapped to an active virtual network.

3.4.5 Mapping communication services into user equipment

The mapping levels formerly described are related to the first phase of the life cycle of a slice [7]. This phase, called Preparation, deals with the design of each previous mapping level, as well as the pre-provisioning and network environment preparation. The second phase consists of installing all slices on top of a physical network and activating them. Then, a mapping of Communication Services into UEs must be done as soon as a new request arrives. Next, in the Run-time phase, all slices are already activated and each UE must be connected to the right CS. More technically, this mapping is essentially a mapping of UE into the right NSI. This is done after a registration procedure by the first AMF receiving the connection request. This function then interacts with NSSF. Based on the orchestrator's policy rules, NSSF is responsible to direct each UE to the right Communication Service. Hence, this last mapping level is essentially concentrated on NSSF, which must be connected to all AMF of other slices offered by CS Providers.

Clearly, a higher level of mapping can only be done if all lower ones are already concluded, creating a strong dependency between them. Additionally, each level in this nested arrangement has its own complexity, and choosing the right mapping to offer a CS request is crucial. Hence, a trade-off between flexibility and readiness is imposed. Deploying Communication Services from the first level gives more flexibility to adapt a virtual environment to a new 5G CS with specific and unexpected technical constraints. However, this might be a very arduous work to be done for every single request. Along these lines, for those highly expected classes of CSs with well-known technical constraints, creating templates to be available to a higher mapping level speeds up the procedure to build a virtualized 5G CS using network slicing techniques.

We have to stress that the decomposition of NSs into NSSs and of NFs into NFSs is, on the one hand, motivated by scalability and efficiency reasons and, on the other hand, requires the network slice design process to take into consideration continuity constraints. Indeed, one NS can be geographically deployed in a scalable manner thanks to the segmentation of a slice into multiple NSSs; and the

overall computing demand can be decreased by allocating resources to NF micro-services rather than to macro NF units. Moreover, depending on techno-economic and network management policies, continuity constraints on the decomposition may be needed; more precisely, the slice provider might decide whether all NSSs belonging to the same higher level NS should undergo the same functional split setting in the Radio Access Network (RAN).

3.5 Sharing policies

The mapping problem rising in 5G described above appears as a novel multi-level embedding of network nodes. Jointly, addressing each mapping strategy involves a crucial point: how will each entity of the same level interact with the higher level instances? Isolation is a key aspect for network slicing and dedicated functions, for example, might be necessary to ensure that each slice operates independently, preventing the incorrect balance of resources between the serving slices. Additionally, security is another extremely important point in virtual environments. To ensure security and data routing control, partially or completely isolated slices with dedicated functions might be implemented.

On the other hand, sharing a function is an interesting strategy to simplify virtual environment implementation and to reduce redundancies throughout the network. Due to the high programmability of NFs, some of them can be installed on top of physical resources to be shared by a set of slices. In addition, following 3GPP standards for 5GC, some NF instances must be common and shared between different slices. This imposition can be interestingly exemplified with NSSF. This function is

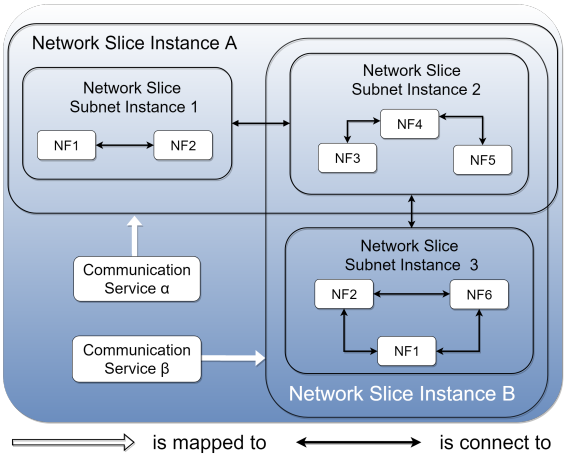


Figure 3.8 – Interactions between Slice and Slice Subnet Instances.

responsible for mapping each UE to the right slice serving a requested CS. In this way, deployed slices must share an interface with NSSF to keep it capable of making the right slice targeting.

It is important to keep in mind that, as functions, an NSSI can also be shared by two or more NSIs, or it might still be entirely dedicated to a slice. Fig. 3.8 illustrates this concept and depicts a possible mapping for different Communication Services. Note that service α is serviced by NSI A, while service β is mapped to NSI B. In this example, NSSI 2 is a shared entity and is serving both NSI A and NSI B. From a holistic point of view, NSSI 2 can be interpreted as an access network serving two cores (NSSI 1 and NSSI 3), or the other way around, for example.

Given the expected data volume from each UE connected to each antenna and the processing capacity of each NF, it is important to predict how many instances of each function type should be installed for each network slice. Moreover, dimensioning strategies have to model how NFs relate to different slices.

Isolation is a key aspect for network slicing and dedicated NFs might be necessary to ensure that each NS operates independently. This approach is important for preventing the incorrect balance of resources between the served NSs. Additionally, security is another crucial point in virtual environments. To ensure security and data routing control, partially or completely isolated network slices with dedicated NFs might be implemented. Hence, isolation constraints might be applied on the virtual layer; NFs installed in the same physical node must be dedicated to a virtual network serving a specific client, thus cannot be shared by two or more NSs. On the other hand, sharing NFs among different NSs can be an interesting strategy to simplify the virtual environment implementation and to reduce redundancies throughout the network [7]. We assume that an NF can treat data from two or more NSs if and only if they have an affinity for each other. By affinity, we mean allowing a network slice to share one or more NFs with another NS. It is important to mention that an NS request might impose isolation constraints only on a specific subset of network functions that cannot be shared with a specific subset of NSs; this might be the case for critical NFs or network slices belonging to the different tenants, for example.

We depict in Fig. 3.9 six possible NF sharing policies that, based on our analysis, are possible as of 3GPP specifications; in this illustration, a DP block can refer to data-plane functions for both access and core segments. They are as follows:

1. *Flat Sharing*: all CSs share the same virtual network; it can be an interesting strategy when different slices have no isolation constraints and show similar technical constraints in terms of latency and availability.

2. *Hard Isolation*: the isolation is complete, each CS has its own virtual network.

3. *Shared Control-Plane*: slices share the same Control-Plane (CP) while having their own and dedicated user Data-Planes (D-DPs); it may be a solution for NSs requiring low end-to-end latency, and in this scenario, DP equipment should be deployed as close as possible to UEs, which has, therefore, an impact on the level of functional splitting.

4. *Partial Control-Plane Isolation*: only a part of the CP, called common CP (C-CP), is shared by two CSs; a CP portion and entire DPs of each CS are dedicated.

5. *Shared Data-Plane*: CSs share the same Data-Plane while having their own and dedicated Control-Planes (D-CPs).

6. *Partial Data-Plane Isolation* case: only a part of the DP is shared by two CSs, named common DP (C-DP); a DP portion and entire CPs of each CS are dedicated.

According to 3GPP specifications, these settings are in practice adaptable to multiple CSs. In addition, regarding the orchestration complexity inherent to each sharing policy (e.g., security and route control), other configurations might be proposed to guarantee Service Level Agreements.

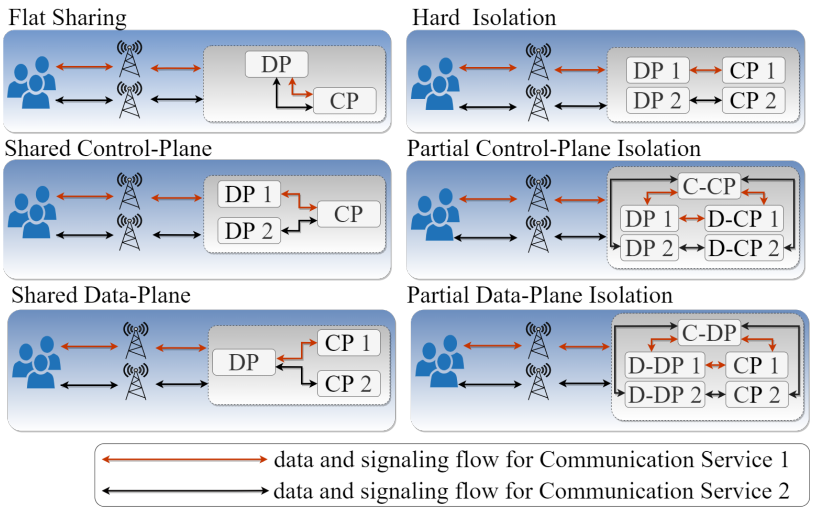


Figure 3.9 – NF sharing policies.

3.6 Challenges

3GPP has been proposing a series of evolution in both Access and Core Networks to better serve the upcoming 5G network. We summarized this work in this chapter with a particular view on function decomposition and related mappings, which seems to call for even more integration of SDN, NFV, and Network Slicing subsystems than what was commonly anticipated. Indeed, new challenges are imposed to operators to ensure that future virtualized networks are successfully operational.

One of the main challenges is related to the first level of mapping. Managing the granularity of a function is a crucial point to minimize the use of physical resources. As a consequence, an intelligent entity is needed to provide the exact sub-set of NF services.What level this granularity is managed and which entity is responsible for it remain both open questions. In other words, will the dynamic creation of customized NFs be in charge of the NFs themselves, the orchestration entity or there will be a new NF dedicated to this end? A similar problem was addressed by [83]. In this work, the authors propose a flexible service composition in order to minimize the redundant functionality found in instances of different service functions (e.g. application and edge firewalls, load balancer, and proxy) throughout the virtual architecture. For this purpose, they apply the microservice concept on service function chains for re-architecting the NFV ecosystem. This interesting approach is directly related to the concept of NF services introduced by 3GPP, however, there is no study proposing such a technique in functions related to 5G Core Network. Hence, this remains an open problem to be addressed.

Furthermore, for each new and uncontemplated use case, new mappings at each level must be performed, causing higher levels to depend intrinsically on the lower levels. In other words, to map slice templates to CSs, for example, a complete function mapping work into slice must already be completed. Clearly, mapping strategies have a strong impact on slice deployment and how to model it. First, an intelligent framework to correctly distribute the flow of data throughout slices must be provided. This is due to the proposed scalability for critical functions to handle highly dynamic data rates. In this context, defining the number of instances for each function within a slice is not a trivial work. Such a framework must be aware of the current state of the network to decide the exact time of scaling up and down some critical NF instances and to redistribute the flow among them in an intelligent manner. [84] presents an approach to address this problem, however, the authors apply such framework only on one critical network function and do not give any instruction on how it could

be implemented on two or more core functions at the same time. In fact, proposing such a global framework is not a trivial task and remains an open problem to be addressed.

All these challenges must be overcome wisely and effectively, as the state of the system can change every second. In legacy technologies, such as 3G and 4G, the entire network system was designed for approximately ten years of use, with small variations and slow evolutions over the period. Contrarily, the 5G system is meant to be extremely flexible and still be able to offer a customized and complete virtual network in minutes for each Communication Service request. In this way, network designers and engineers have a laborious road to develop and apply techniques, algorithms, models, and approaches that guarantee the high efficiency in translating Communication Service demands into slices, doing all the necessary mappings, allocating physical and virtual resources, and activating, monitoring, and evolving each slice on 5G system.

Chapter 4

Device-to-device communication in radio access networks

Contenu

In future 5G networks, mobile User Equipment (UEs) will be able to host functions that give them new abilities such as sharing connectivity, capacity, and CPU resources with other UEs, regardless of the ongoing traditional communications. The 5G wireless technology, along with the evolution of mobile users' behavior and needs, will make the current scheme of communication (UE to Base Station) no longer optimal in terms of radio resource utilization. The Device-to-Device (D2D) communication mode is one of the new approaches presented as a promising alternative to traditional communication in cellular networks. D2D communication is defined as a direct communication between two mobile or fixed user devices, without traversing the Base Station (BS) [65]. This technology allows to reuse

radio resources and to decrease the end-to-end latency of local communications. Then, D2D would allow a set of UEs geographically close to each other to establish direct D2D communications, or span multiple links (multi-hop D2D communications), to access a given service (e.g. video streaming or gaming) while ensuring the required service quality.

A *domain* is defined as the set of UEs and BSs that are used to establish mobile communications (D2D or cellular) related to a specific service. The communication is either direct or uses multiple links (D2D or via the BS). Two UEs can then communicate through cellular links, using the BS or D2D links, and both technologies can coexist within the same mobile network. In any case, radio resources should be allocated to every active link involved in a communication, and the SINR (Signal-to-Interference-plus-Noise Ratio) level required by the service should be ensured.

In this chapter, we formally define the Domain Creation Problem and we propose a node-arc (compact) ILP formulation to model it. We present some strategies to enhance the linear relaxation of the formulation along with two classes of valid inequalities. Our results are embedded within a branch-and-cut algorithm to solve the problem. Numerical experiments are made on instances generated thanks to realistic parameters of Orange mobile networks[1].

4.1 Problem statement

We consider a mobile network composed of a set of devices (UEs), a set of antennas (BS), and a set of services eligible to D2D communications, with their associated traffic matrices. These traffic matrices are in the form of data volume to be exchanged between pairs of devices. The involved devices can communicate through one or several links, either using D2D or cellular communication (via the BS). A non-negative weight, corresponding to the SINR, is associated with each link. It is a measure of the quality of the communication that could be established using this link. Every service requires a minimal quality threshold in terms of SINR and available resources (hardware capacity for the devices, radio resources for the links) to be successfully established. Once these conditions are met, the services are delivered through the different types of resources allocated to both the devices and the links.

1. The content of this chapter was published in the following paper:
Benhamiche, Amal, Wesley da Silva Coelho, and Nancy Perrot. "Routing and Resource Assignment Problems in Future 5G Radio Access Networks." International Network Optimization Conference. 2019.

The network is represented by a directed graph $G = (V \cup U,\ A)$, where V is the set of nodes associated with devices, U the set of Base Station nodes, and A the set of arcs. We denote by $\delta^+(u)$ (resp. $\delta^-(u)$) the subset of arcs going from (resp. to) node u. Every node $u \in V$ has an associated weight vector $c^u = \{c^u_1, \ldots, c^u_{|C^d|}\}$, where $c^u_i \geqslant 0$ is the capacity available at node u for the physical resource $i \in C^d$. Every arc $e \in A$ has a weight denoted $SINR_e$ that expresses a measure on the quality of the transmission link represented by e. For every pair of arcs $e, f \in A$ we denote by $d(e, f)$ the *distance* between e and f. This value corresponds to the minimum distance between the opposite ends of the given pair of links, that is, the origin of one and the destination of the other. Namely, two arcs e and f are said to be *close* if $d(e, f) \leqslant D$, where D is a minimum acceptable distance. Let $R = \{1, \ldots, r\}$ be the set of available radio resources. An arc e of A is said to be *active* if a radio resource $r \in R$ is assigned to it. A resource $r \in R$ can be assigned to two different arcs $e, f \in A$ only if $d(e, f) > D$. Indeed, due to interference constraints, two communications cannot be established on e and f simultaneously using the same resource $r \in R$ if e is close to f. We denote by K the set of traffic demands to be routed and S the set of service types. Every demand $k \in K$ is defined by an origin node $o^k \in V$, a destination node $d^k \in V$ and a requested service s_k. Moreover, every $k \in K$ has an associated cost $\mu^e_{s_k}$ to use arc $e \in A$ and a traffic vector $a^{s_k} = (a^{s_k}_1, \ldots, a^{s_k}_m)$ where the element $a^{s_k}_m \geqslant 0$ denotes the quantity of physical resource type m from the set C^d of all resources types (e.g. CPU, RAM, storage) needed to process the service s_k requested by k. Finally, we denote by β_k the quality threshold needed by k to access the required service s_k.

In this context, we define the Domain Creation Problem (DCP) as follows.

Definition 4.1.1. *The Domain Creation Problem consists in finding a minimum cost allocation of the radio resources in R to the active arcs of G to provide a feasible routing path for each demand. In particular, a routing $p^k = \{(o^k, u), \ldots, (v, d^k)\}$ for a demand k is said to be feasible if*

- *all the arcs of p^k have an SINR value above the quality threshold β_k required by the demand k and,*
- *all the nodes in p^k have enough capacity to satisfy the resource requirements of k.*

Example An instance of the problem, with fives devices and one BS, is illustrated in Figure 4.1. For sake of clarity, each pair of arcs between two nodes is represented by an edge. The edges representing D2D links are shown in solid lines while edges representing device to BS links are in dashed lines. Two different services, namely *gaming* and *video streaming*, and three demands per service are to be

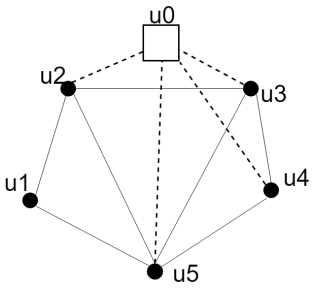


Figure 4.1 – Example of a network with 6 nodes and 6 demands to be delivered for 2 different services

Figure 4.2 – Feasible solution: active links and associated radio resource for *Gaming* domain (left) and *Video streaming* domain (right)

delivered:$\{(u_1,u_3),\ (u_2,u_5),\ (u_4,u_2)\}$ and $\{(u_4,\ u_3),\ (u_3,\ u_1),\ (u_5,\ u_2)\}$, assuming that twelve radio resources $\{r_1,\ldots,r_{12}\}$, are available. A feasible solution is represented in Figure 4.2. The figure on the left side represents the *Gaming* domain, where all three demands are satisfied through D2D links. The figure on the right side is the *Video streaming* domain, where just one demand uses the BS. Note that, using the legacy approach, all demands must pass through the BS, using one uplink (from UE origin to BS) and one downlink (from the BS to UE destination) for each demand. Since all active links share at least one arc extremity (BS), the solution would require all the available resources (a different one for each active link) to avoid interference. Thus, using D2D communications allow here to save 50% of radio resources compared to a "fully cellular" solution.

4.2 Related works

The problem of assigning radio resources to transmission links is studied in [85] and [86] under the denomination of Frequency Assignment Problem (FAP). In [85], the authors give an ILP formulation for the problem and propose a branch-and-cut algorithm to solve it. The work in [86] presents sev-

eral variants of FAP and discusses existing and original optimization (including exact and heuristic) approaches to solve them. The routing and resource allocation aspects are combined in the so-called *Routing and Wavelength Assignment* (RWA) problem (see [87] and [88] for instance) which arises in Optical Networks. The DCP differs from the above-cited problems in that traffic demands are *unsplittable* (each demand has to be sent along a unique path using D2D or cellular links), several types of capacities on the devices are considered, and radio resources re-use is submitted to distance constraints to avoid radio interference. [89] propose a greedy heuristic as an alternative to allocate resources to active links using D2D communication. The proposed heuristic is divided into two phases, each one responsible for the allocation of uplinks and downlinks. After several tests in realistic scenarios, the authors showed that the greedy heuristic improves the total throughput significantly reducing the interference between classic and D2D communications. To maximize the number of active links respecting minimum quality requirements in wireless networks, authors in [90] proposed a new integer programming algorithm based on an effective representation of the SINR constraints which is inspired by knapsack problems using lifting procedure.

4.3 A mathematical formulation for the DCP

In this section, we propose a compact ILP formulation for the DCP followed by some valid inequalities to be used in a branch-and-cut algorithm.

4.3.1 Notations and formulation

The three types of binary variables are:

• x_{er}^k, $e \in A$, $k \in K$, $r \in R$ that takes the value 1 if the arc e is used by the demand k and assigned with the resource r, 0 otherwise.

• y_i^k, $i \in V \cup U$, $k \in K$, that takes the value 1 if the node i is used by the demand k, 0 otherwise.

• z^r, $r \in R$ that takes the value 1 if the resource $r \in R$ is assigned with at least one arc, 0 otherwise.

Then, the DCP can be formulated as:

$$\min \quad \sum_{k \in K} \sum_{e \in A} \sum_{r \in R} \mu_e^{s_k} x_{er}^k + \sum_{r \in R} \psi z^r \tag{4.1}$$

s.t.

$$\sum_{e\in\delta^-(u)}\sum_{r\in R} x_{er}^k - \sum_{e\in\delta^+(u)}\sum_{r\in R} x_{er}^k = \begin{cases} 1, & \text{if } u = d_k, \\ -1, & \text{if } u = o_k, \\ 0, & \text{otherwise,} \end{cases} \quad \forall k \in K, \forall u \in U \cup V \tag{4.2}$$

$$x_{er}^k \beta_k \leq SINR_e \qquad \forall\ k \in K, \forall\ e \in A, \forall\ r \in R \tag{4.3}$$

$$\sum_{k\in K\setminus T(i)} y_i^k a_m^{s_k} \leq c_m^i \qquad \forall\ i \in V, \forall\ m \in C^d \tag{4.4}$$

$$2x_{er}^k - y_i^k - y_j^k \leq 0 \qquad \forall k \in K, \forall r \in R \forall (i,j) = e \in A \tag{4.5}$$

$$x_{er}^k + x_{fr}^{k'} \leq z^r \qquad \forall r \in R, \forall k, k' \in K, \forall e \in A, \forall f \in D(e) \tag{4.6}$$

$$x_{er}^k \in \{0,1\} \qquad \forall k \in K, \forall e \in A, \forall r \in R \tag{4.7}$$

$$y_i^k \in \{0,1\} \qquad \forall k \in K, \forall i \in V \tag{4.8}$$

$$z^r \in \{0,1\} \qquad \forall r \in R \tag{4.9}$$

This formulation has a polynomial number of variables and constraints. The objective (4.1) is to minimize the total costs composed of non-negative routing and radio resource utilization costs. The first set of inequalities (4.2) are the flow conservation constraints. They ensure that each demand is routed along a unique path between its origin node and its destination node. Note that such a routing path can either span arcs corresponding to D2D links or use a node of U corresponding to some BS. Inequalities (4.3) guarantee that a demand for a given service is routed along edges whose SINR satisfies the quality threshold required by this service. (4.4) express the capacity constraints in every node for the different types of hardware resources, with $T(i)$ being the set of demands $k \in K$ that have $i \in V$ as origin or destination. These capacity constraints are needed only on intermediate nodes, that is, the nodes that are not the origin nor the destination of a given demand $k \in K$. Inequalities (4.5) are linking constraints and inequalities (4.6) guarantee that the same radio resource is not assigned to different edges unless they are distant enough, where $D(e)$ is a set of arcs that are close to a given arc $e \in A$. Note that, as the objective is to minimize the resources, in any optimal solution one radio resource at most will be allocated to the same arc for a given demand. Finally, (4.7)-(4.9) are the integrality constraints.

The ILP formulation (4.1)-(4.9) can be strengthened by replacing inequalities (4.3) by:

$$x_{er}^k \leqslant \lfloor \frac{SINR_e}{\beta_k} \rfloor, \forall k \in K, e \in A, r \in R. \tag{4.10}$$

4.3.2 Symmetry-breaking

Due to the inherent symmetry of this problem, there is possibly a large number of feasible solutions.

$$(x, y, z) \in \{0,1\}^{(m\times|R|+n)\times|K|+|R|}$$

satisfying inequalities (4.2)-(4.6), where $m = |A|$ and $n = |V|$. The breaking symmetry constraints (4.11) are inspired by classical inequalities in combinatorial optimization (see [91]), and can help in reducing the number of symmetric solutions in the formulation.

$$z^r \geqslant z^{r+1}, \qquad \forall 1 \leqslant r \leqslant |R| - 1, \tag{4.11}$$

(4.11) allow to assign the radio resources in an ordered way, forbidding to use a resource $r+1$ if r is available. Note that a vector $(\overline{x}, \overline{y}, \overline{z}) \in \{(x, y, z) \in \{0,1\}^{(m\times|R|+n)\times|K|+|R|} : (x, y, z)$ satisfies (4.2) $-$ (4.11)$\}$ is clearly a feasible solution to the DCP.

4.3.3 Valid inequalities

We introduce now two classes of inequalities valid for the DCP.

4.3.3.1 Clique-based inequalities

Given an instance of DCP, we define the *conflict graph* associated with a node $u \in V$ as follows. For each capacity type $m \in C^d$, let $\mathcal{H}_u^m = (V_u^m, E_u^m)$ be the undirected graph obtained from the set of demands K as follows. A node v_k in V_u^m is associated with every demand $k \in K$ and there exists an edge $v_k v_l \in E_u^m$ between two nodes v_k, v_l of V_u^m if $a_m^{s_k} + a_m^{s_l} > c_m^u$. In other words, an edge in $\mathcal{H}_u^m$ exists if two demands cannot be packed together in the node u due to the lack of capacity for the resource type m. Consequently, a clique in the graph $\mathcal{H}_u^m$ corresponds to a set of demands that cannot use simultaneously the node u. Hence, we denote by $\mathcal{C}_\mathcal{H}$ the set of cliques in the graph $\mathcal{H}_u^m$. Let u be a node of V and $m \in C^d$ be a physical resource type. Then the following inequalities

$$\sum_{k\in\mathcal{C}} y_u^k \leqslant 1, \forall \mathcal{C} \in \mathcal{C}_\mathcal{H} \tag{4.12}$$

are valid for the DCP.

Proof. Let $\widetilde{\mathcal{C}}$ be a clique in $\mathcal{H}_u^m$. It is clear that if two demands k_1, k_2 from clique $\widetilde{\mathcal{C}}$ use the resource m of node u, the capacity constraint (4.4) for the resource m will be violated. In other words, each edge $v_{k_i}v_{k_j}$ of the clique $\widetilde{\mathcal{C}}$ represents an infeasible packing of the demands k_i, k_j in the node u. □

4.3.3.2 Strengthened neighborhood inequalities

The second family of valid inequalities strengthen constraints (4.6). They are obtained by considering the *interference* graph $\mathcal{N} = (V_N, E_N)$ defined as follows. Every node $u \in V_N$ corresponds to an arc in A and two nodes u_e, u_f of V_N (associated respectively with the arcs e and f from A) are interconnected by an edge if e and f are close enough from each other (ie if $d(e, f) \leqslant D$). Consequently, a clique in the graph $\mathcal{N}$ corresponds to a subset of arcs in A that are pairwise close, and cannot receive the same radio resource due to interference constraints. Likewise in the conflict graph defined before, we denote by $\mathcal{C}_\mathcal{N}$ the set of cliques in the graph $\mathcal{N}$. The following inequalities

$$\sum_{k\in K}\sum_{e\in\mathcal{C}} x_{er}^k \leqslant z^r, \forall r \in R, \mathcal{C} \subseteq \mathcal{C}_\mathcal{N} \tag{4.13}$$

are valid for the DCP.

Proof. Let $\widetilde{\mathcal{C}}$ be a clique in $\mathcal{N}$ and u_e, u_f two nodes of V_N that belong to clique $\widetilde{\mathcal{C}}$. Clearly, if e and f are allocated the same resource $r \in R$ in a solution, then it cannot be feasible for the DCP. □

Note that similar inequalities are used in [85] and [86] for the Frequency Assignment Problem.

In what follows, we describe our branch-and-cut framework and show some computational results to assess the efficiency of our cuts.

4.4 A branch-and-cut algorithm for the DCP

We developed a branch-and-cut algorithm for the DCP based on two heuristic procedures to generate dynamically inequalities (4.12) and (4.13). Both separation routines rely on a greedy algorithm introduced in [92] for the Independent Set problem, that finds a clique in the conflict graph (respectively the interference graph) with appropriate weights on the nodes. We then add the corresponding violated clique-based (respectively strengthened neighborhood) inequalities, if any, to the current LP. Both classes of valid inequalities are separated throughout the branch-and-cut tree and several inequalities may be added at each iteration of the algorithm.

4.5 A 2-phase heuristic for the DCP

Since solving the initial formulation (4.1)-(4.9) presented in the previous chapter has impractical runtime even for small instances, we propose a solving method based on a decomposition of the DCP into two sub-problems: the *routing sub-problem* and the *resource allocation sub-problem*, that are to be solved separately. The objective of the first sub-problem is to find an elementary path for each demand while minimizing the total link utilization costs. Then, the second sub-problem provides a resource allocation to each active link obtained from the routing sub-problem solution.

We further propose a two-phase heuristic, obtained by decomposing the problem into routing and radio resource allocation sub-problems. To solve the routing sub-problem, we propose two methods: an LP-based heuristic from the linear relaxation of a compact formulation, and a non-compact formulation obtained by generating a subset of relevant paths. Then, the allocation sub-problem is transformed into a vertex coloring problem that is solved heuristically by an improved greedy algorithm. This greedy algorithm is compared to a dual bound given by the exact solution of the associated Max-Clique Problem. In what follows, we present in detail each of the proposed approaches.

4.5.1 Routing sub-problem

We propose two formulations for the routing sub-problem: a compact formulation obtained by relaxing the resource assignment constraints (4.6) from (4.1)-(4.9), and a path reformulation.

4.5.1.1 Compact formulation

This formulation is the compact formulation obtained by relaxing the resource allocation constraints (4.6) from the formulation (4.1)-(4.9). The returned solution is a set of elementary paths for each request, respecting the capacities of the nodes along the paths. Two kinds of binary variables remains in the formulation : x_e^k that takes value 1 if the link $e \in A$ is used by the request $k \in K$ and the variables y_k^i. The objective is to minimize the total cost of active links:

$$min \quad \sum_{k \in K} \sum_{e \in E} \mu_e^{s_k} x_e^k \tag{4.14}$$

Solving approach: The linear relaxation of this formulation is strengthened replacing (4.5) by:

$$x_e^k - y_i^k \leq 0 \qquad \forall\ k \in K, \forall\ (i,j) = e \in E, \tag{4.15}$$

$$x_e^k - y_j^k \leq 0 \qquad \forall\ k \in K, \forall\ (i,j) = e \in E. \tag{4.16}$$

and adding the following inequalities :

$$y_i^k (a_m^{s_k} - c_m^i) \leq 0 \qquad \forall\ i \in V, \forall\ m \in C^d, \forall\ k \in K \backslash T(i). \tag{4.17}$$

The linear relaxation of this strengthened sub-problem is then solved and a heuristic procedure is used to get a feasible integer solution. This approach is summarized in Algorithm 1.

Algorithm 1: LP-based Heuristic for the Routing sub-problem

```
1: Solve LP (G, K)
2: Let FractionalDemands be the set of demands with associated optimal fractional variables
3: for each demand k do
4:    if all associated variables have integer value then
5:       update the LP by decreasing used capacities on vertices traversed by the associated paths
6:    else
7:       FractionalDemands : FractionalDemands ∪{k}
8:    end if
9: end for
10: Solve ILP (G,FractionalDemands)
11: Update solution
12: Return solution found =0
```

First, the linear relaxation is solved to optimality. Then, variables having an integer optimal value are fixed by updating the right-hand side of the constraints (4.5) in step 5. Finally, this residual ILP formulation is solved to optimality, giving rise to an integer solution for the whole problem. Step 11 is the rounding procedure on the solution found by step 1. This heuristic is particularly efficient for this problem since most of the optimal variable values of the linear relaxation are integers.

4.5.1.2 Path formulation

The second formulation is a path formulation. We assume that all feasible paths P_k have been previously generated for each demand k. Thus, we define new binary variables x_p^k that take value 1 if the path $p \in P_k$ is used by the demand $k \in K$, 0 otherwise. The objective now is to minimize the sum of the active path weights:

$$min \quad \sum_{k \in K} \sum_{p \in P_k} \mu_p^{s_k} x_p^k \tag{4.18}$$

where $\mu_p^{s_k}$ is the cost of path $p \in P_k$, defined as the sum of the weights on the arcs of p. Let α_p^e, respectively γ_p^i, be an indicator parameter with value 1 if the arc e, respectively the node i, is in the path p. The constraints of the path formulation are:

$$\sum_{p \in P_k} x_p^k = 1 \qquad \forall\, k \in K, \tag{4.19}$$

$$\alpha_p^e x_p^k \beta_{s_k} \leq sinr_e \qquad \forall\, k \in K, e \in A, \forall\, p \in P_k, \tag{4.20}$$

$$\sum_{k \in K \setminus T(i)} \sum_{p \in P_k} \gamma_p^i x_p^k a_m^{s_k} \leq c_m^i \qquad \forall\, i \in V, \forall\, m \in C^d, \tag{4.21}$$

$$x_p^k \in \{0, 1\} \qquad \forall\, k \in K, \forall\, p \in P_k. \tag{4.22}$$

Constraints (4.19) assure that each demand uses one path. Constraints (4.20) are the SINR constraints, and (4.21) are the capacity constraints.

Solving approach: This formulation, restricted to a subset of paths, is solved to optimality. The subsets of paths are generated by applying the algorithm proposed by Yen [23]. This algorithm returns the K-shortest loopless paths for a graph with non-negative edge costs. It uses a shortest path algorithm as an intermediary to construct the whole solution. In this work, we use Dijkstra's algorithm [19], which has a good performance and a polynomial complexity. Using Yen's algorithm gives a guarantee to generate all the feasible paths due to the characteristics of our use case: for each demand, once a path uses the base station we are sure that all the feasible D2D paths have already been generated, then the path generation is stopped. This feature is due to greater weights on BS arcs. The difference between the weights of the BS links and the D2D ones plays an important role in the construction of the paths subsets: the greater this difference is, the longer it could be to obtain all the paths that use only D2D communication. Finally, to reduce the size of the formulation, a pre-processing on the SINR constraints is operated before solving both routing formulations: only the valid SINR link constraints are kept in the formulation.

4.5.2 Resource allocation sub-problem

The resource allocation sub-problem consists in allocating the radio resources to each active link provided by the solution of the first sub-problem. For this purpose, let $G^a(V^a, E^a)$ be a graph where each vertex in the set V^a represents an active link, that is, a link used by at least one path of the routing problem solution (steps 2-5 in Algorithm 2). An edge $e \in E^a$ is associated with a pair of

vertices if the corresponding active links cannot share the same radio resource due to interference (steps 6-12). The objective is to assign a minimum number of colors (resources) to the vertices of the graph G^a, in such a way there are no adjacent vertices with the same color (step 14). This falls into a classical Vertex Coloring Problem [93].

Algorithm 2: Resource Allocation sub-problem

1: Set V^a and E^a to empty sets
2: **for** each activated link $e \in A$ **do**
3: Set Vertex $auxVertex.id \longleftarrow e.id$
4: Set $V^a : V^a \cup auxVertex$
5: **end for**
6: **for** each pair $(u, v) \in V^a$ **do**
7: **if** $dist(u, v) \leq criteria$ **then**
8: Set Link $auxLink.extremity1 \longleftarrow u$
9: Set Link $auxLink.extremity2 \longleftarrow v$
10: Set $E^a : E^a \cup auxLink$
11: **end if**
12: **end for**
13: Set graph $G^a(V^a, E^a)$
14: Coloring (G^a)
15: Return Resource Assignment =0

The proposed heuristic has as input the initial graph of the problem, the values of the routing solutions, and a criteria value that represents the minimum distance for a pair of links to have the same resource. This value was fixed to $D = 100$ meters to be representative of realistic use cases. Hence, for $dist()$ method, we calculate the distance between the opposite ends of the pair of links, that is, the origin of one with the destination of the other. The method returns *zero* for adjacent links. Finally, to find the coloring of the graph G^a (that is, allocate the resources to each active link) we use the $Coloring()$ method which is the implementation of a classical greedy algorithm [94] for the Vertex Coloring Problem.

It is well known that the performance of this greedy algorithm is sensitive to the order of choice of the next vertex to be colored. For this reason, we randomly generate a large number of orders and choose the one that returns the minimum amount of colors. A lower bound value for the optimal solution is given by solving the associated Max-clique problem. For this purpose, we use the method proposed by [95], which is an exact approach using parallel programming.

Table 4.1 – Runtime comparison (in seconds) with strengthening constraints.

Instance	Original Model	Symmetry-Breaking	SINR	Symmetry & SINR
U5_D2	0.05	0.06	0.05	0.06
U5_D6	187.18	11.55	77.18	**10.71**
U5_D8	600.56	**45.16**	90.70	72.43
U10_D5	1677.21	**290.33**	697.49	435.08
U10_D7	8746.32	3569.18	4453.58	**2967.45**
U15_D7	10800	10800	10800	10800

4.6 Computational results

We now present the numerical experiments applying each of the proposed approaches to solve the DCP. We first discuss the efficiency of the branch-and-cut algorithm and the symmetry-breaking constraints. Then, we analyze the numerical simulations applying our 2-phase algorithm on different realistic instances.

4.6.1 Branch-and-cut and symmetry-breaking constraints

We present below some experiments obtained for a set of small instances containing 5 to 15 nodes and up to 7 demands. For these tests, each scenario contains only 1 antenna and 2 service domains. For each instance, realistic data was provided by Orange, including the network topologies and SINR values. We implemented our approach in a C++ environment using ILO CPLEX 12.6 as the linear solver. Our tests were run on a Linux server with an Intel Xeon E5-2650 CPU.

Table 4.1 shows the impact of inequalities (4.10) and (4.11) on the initial model (formulation (4.1)-(4.9)). In the first column, the name of each instance refers to the number of users (U#) and demands (D#). The next four columns show the computational time (in seconds) for the initial formulation, then when adding symmetry breaking constraints (4.11), strenghthened SINR constraints (4.10) and both constraints, respectively. We can notice that using constraints (4.11) allows to obtain the best execution time for two out of five instances tested ($U5\_D8$ and $U10\_D5$) while the instances $U5\_D6$ and $U10\_D7$ have the lower runtime when applying constraints (4.10) and (4.11), combined. For instance $U15\_D7$, the optimal solution could not be found after 3 hours of execution.

We tested the impact of using our valid inequalities and compared the results to CPLEX branch-and-bound for the strengthened model (formulation (4.1)-(4.2) + (4.4)-(4.11)). Table 4.2 shows the results obtained on five instances (same as in Table 4.1) when (i) no additional cuts are used, (ii)

Table 4.2 – Solution quality comparison between models with and without additional cuts.

Instance	Strengthened Formulation			Strengthened + Neighborhood cuts		
	root gap (%)	runtime (s)	tree size	root gap (%)	runtime (s)	tree size
U5_D2	**0.00**	0.05	**1**	**0.00**	0.06	**1**
U5_D6	10.00	11.49	7	**10.00**	11.67	8
U5_D8	59.65	154.65	470	**23.58**	130.69	**291**
U10_D5	57.89	677.21	1327	**33.09**	1370.90	190
U10_D7	62.26	3236.61	1850	**50.04**	8654.21	2040
*U15_D7**	77.23	10800	58	**72.37**	10800	8
Instance	Strengthened + Clique-based cuts			Strengthened + both cuts		
	root gap (%)	runtime (s)	tree size	root gap (%)	runtime (s)	tree size
U5_D2	**0.00**	0.06	**1**	**0.00**	0.05	**1**
U5_D6	**10.00**	12.33	**2**	**10.00**	13.31	**2**
U5_D8	33.71	389.01	1225	32.00	150	599
U10_D5	48.23	1262.83	**91**	33.09	2265.83	**84**
U10_D7	62.26	3251.99	1850	50.04	9648.12	2040
*U15_D7**	77.23	10800	70	72.37	10800	22

using strenghthened neighborhood inequalities (4.13) in addition to formulation (4.1)-(4.9), (*iii*) using clique-based inequalities (4.12) in addition to strengthened model and (*iv*) both cuts are used in the branch-and-cut. We can observe that the gap at root node is substantially reduced when adding cuts (the gap value decreases from 59.65% to 23.58% for instance $U5\_D8$ when using strenghthened neighborhood cuts) and so for the size of the branch-and-cut tree (for instance $U10\_D5$, we range from 1327 nodes without cuts to 84 nodes when both cuts are used). Overall, the strenghthened neighborhood cuts are more efficient in reinforcing the strengthened model.

4.6.2 2-phase heuristic

We present below some experiments obtained from applying the proposed 2-phase heuristic on a set of different instance sizes. We implemented our approaches in a C++ environment using ILO CPLEX 12.6 as the linear solver. Our tests were run on a Linux server with an Intel Xeon E5-2650 CPU. The solver's time limit was set to 7200 seconds (2 hours) and only one thread was provided to the branch-and-bound process.

4.6.2.1 Test setup

All instances were generated and offered by the Mobile Access Design team from Orange Labs, in which each cell has a 500 meters radius and is based on the networks of the figure 4.3 and follow the following premises:

— The total number of UEs is evenly divided between the 7 core cells.

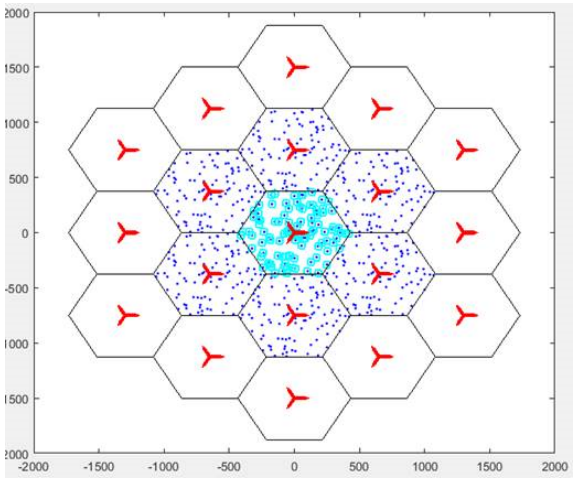


Figure 4.3 – Simulated Network.

— We set the link weight to:
  — 1.0 between each UEs pair in the central cell if the distance between them is less than 100 meters;
  — 1.49 between a UE and the nearest base station;
  — 0.9 between each pair of base stations;
— The origins of all requests are in the central cell;
— The requests are unique, that is, there is only one with the specific configuration (origin, destination, required service), and random values are generated for only three UE physical capacities (e.g., CPU, battery, and RAM).
— 6 different service domains are available to be required.

As previously presented, the name of each instance refers to the number of users (U#), links (L#), and demands (D#).

4.6.2.2 Numerical results

We use this section to present and analyze the results obtained from the numerous tests performed with the instances presented in the previous section. Such tests have as main objective to analyze the performance of each approach of our 2-phase formulations in terms of execution time and quality of the solution obtained in each phase.

Routing sub-problem:

The numerical tests of the routing subproblem are summarized in table 4.3. The first column of the Compact Formulation part is the percentage of *active* variables that *are not* integers in the optimal LP solution. We note that it is always less than 6%. The second column is the gap between the solution found by the relaxation and the final solution found by the LP-based heuristic. The average gap is 2.30% with a standard deviation equals to 3.85%. Finally, the third column is the total run time in seconds (pre-processing, relaxed solution, and LP-based heuristic). In the second part of table 4.3, the pre-processing and solver run times of the path formulation are presented. The average number of paths generated in pre-processing is equal to 5.26 for each demand (with a standard deviation equals to 2.39). Most of the runtime of the compact formulation-based approach was spent in solving the linear relaxation of the problem. Given the very low number of fractional variables in the relaxed solution, the last IP on the residual formulation is quickly solved. It is also worth noticing that the pre-processing in path formulation is the most important step since it is responsible for most of the solving time for all instances. However, its performance is relatively better than for the compact formulation, being on average 1.30 times faster. It is important to mention that the gap between the two formulations was always less than 0.50%.

Resource allocation sub-problem: Table 4.4 shows the results for the resource allocation subproblem, where the *Pre-processing* column represents the time needed to transform the solution from the previous subproblem into a classic graph and find its max-clique. We can observe that even though they are extremely large graphs, the time needed to find the final solution is relatively short. This is due to the characteristics of the topology constructed from the assumptions and hypotheses previously

Table 4.3 – Routing subproblem: numerical simulations.

	Compact formulation			Path Formulation	
Instances	Act. Frac. Var. (%)	Gap (%)	Runtime (s)	Pre-processing (s)	Solver (s)
U700_D175	0.05	0.03	9.16	4.81	0.02
U700_D350	1.92	1.1	13.92	9.5	0.02
U700_D525	1.84	13	21.28	14.26	0.08
U700_D700	0.87	0.04	26.72	19.22	0.13
U1400_D350	5.22	0.02	46.83	27.48	0.8
U1400_D700	3.37	3.6	77.7	55.68	0.26
U1400_D1050	3.13	3.05	152.31	82.82	0.42
U1400_D1400	1.79	0.49	143.78	110.81	0.60
U2100_D525	5.56	0.02	143.28	81.28	0.23
U2100_D1050	3.54	0.09	257.18	163.39	0.65
U2100_D1575	2.15	3.86	242.84	230	0.8

Table 4.4 – Resource Allocation sub-problem: numerical simulations

Instances	Pre-processing (s)	Greedy (s)	Gap (%)
U700_D175	0.13	0.89	0
U700_D350	0.53	1.81	0
U700_D525	1.15	2.75	0
U700_D700	2.12	3.89	0
U1400_D350	0.54	1.74	0
U1400_D700	2.34	3.76	0
U1400_D1050	4.97	5.95	0
U1400_D1400	7.06	6.97	0
U2100_D525	1.32	2.07	0
U2100_D1050	4.45	5.25	0
U2100_D1575	11.21	9.51	0

presented. Generated graphs have an average density equals to 64.05% - standard deviation equals to 1.98%. Another important result emphasizes is that in all cases, we found the optimal solution, proven by the lower bound value previously calculated by the exact max-clique algorithm (last column).

Final solution: In order to analyze the quality of the global solution, we performed additional tests to compare the solutions found by our approach and those found by the original model (4.1)-(4.9). Due to the high complexity of the original model, we run tests with only small instances. Table 4.5 shows the time spent on each approach, as well as the solutions found. Regarding the results applying the proposed 2-phase heuristic, *Runtime* columns represent the total time spent by both pre-processing and solving steps one each approach. As seen in Table 4.5, our approach has a short time calculation and reached the global optimum in all instances in these simulations.

Table 4.5 – Quality of the final solution.

Instance	2-Phase Heuristic				Branch-and-Bound	
	Compact Formulation		Path Formulation		Original Formulation	
	Gap (%)	Runtime (s)	Gap (%)	Runtime (s)	Gap (%)	Runtime (s)
U140_L334_D35	0.0	0.19	0.0	0.15	0.0	7816
U140_L342_D35	0.0	0.19	0.0	0.16	0.0	8523
U140_L346_D35	0.0	0.18	0.0	0.14	0.0	7822
U140_L333_D35	0.0	0.19	0.0	0.16	0.0	7560
U140_L329_D35	0.0	0.17	0.0	0.13	0.0	7987
U140_L336_D35	0.0	0.18	0.0	0.14	0.0	8622
U210_L518_D52	0.0	0.46	0.0	0.43	0.0	32892
U210_L501_D52	0.0	0.43	0.0	0.34	0.0	37808
U210_L512_D52	0.0	0.45	0.0	0.35	0.0	38091
U210_L522_D52	0.0	0.45	0.0	0.39	0.0	33790
U210_L513_D52	0.0	0.43	0.0	0.40	0.0	36987
U210_L508_D52	0.0	0.45	0.0	0.38	0.0	37090

4.7 Summary

In this chapter, we studied the Domain Creation problem, that is a routing and resource assignment problem arising in future 5G networks. We proposed two algorithms: exact and heuristic, to solve it. The exact approach is based on a node-arc ILP formulation enhanced by two families of valid inequalities that are used within a branch-and-cut framework. The preliminary results show a significant impact of the cuts in strenghthening the LP relaxation and reducing the computation time. We expect that adding further classes of cuts and performing an analysis to find out the specificities of difficult instances (regardless of their size) will allow to solve even larger instances. A natural question would be to consider a non-compact formulation, based on path variables and propose a column generation based algorithm to solve it. On an other hand, our experiments show that the heuristic approach performs well, even on large instances with up to 2100 devices and 1500 service requests on 7-cell networks. It would be interesting and most probably very powerful to use it as a primal heuristic to boost efficiency of an exact algorithm.

Chapter 5

The Network Slice Design problem

Contenu

Telecommunications network infrastructures evolved with 5G [96] and the development of the 'network slice' as a novel virtualized infrastructure model. This technology now not only covers application-level slice abstraction as done with preliminary works on 'slicing', but also physical and switching layers virtualization, with different radio access and link communication technologies. This transition challenges slice network design since multiple resources and segments, historically managed independently from each other, are to be operated with continuity in networking and computing resource allocation and provisioning as a whole and unique service. In this context, different providers can be associated with different communication services running on the same physical network at the

access, core, and application segments.

In this chapter, we formally define the Network Slice Design problem (NSDP) as a comprehensive network function dimensioning, placement, routing, and mapping framework that (i) takes into consideration the above mentioned new mapping dimensions and (ii) models the relationship between flexible radio access functional splitting, control-plane and data-plane function isolation, and core network function placement. We use the 3rd Generation Partnership Project (3GPP) [8, 7, 6] 5G as a reference system, knowing that the next generations shall support the same functional requirements in terms of function mapping, sharing, and placement.

Even though several works partially cover the network slice design problem [97, 98] and related sub-problems, such as functional split mode selection [63, 64, 56], network slicing with VNF sharing [99, 100, 101], and network slicing with VNF scaling [84, 102, 103, 104, 105], no attention has been given to address jointly all aforementioned aspects in order to design network slices and understand the impact of mapping, sharing and split policies.

The overall objective of this chapter is thereby to introduce and study the Network Slice Design Problem in 5G systems. We first propose a Mixed-Integer Linear Programming (MILP) formulation for the problem including novel *splitting*, *mapping* and *provisioning* constraints described in the published 5G standards documents [6, 7, 8] and analyze its complexity[1].

5.1 Problem statement and notations

In this section, we introduce a set of necessary notations and we give a formal definition of the Network Slice Design Problem. Table 5.1 summarizes the notations.

5.1.1 Physical network

The physical layer is modeled by a directed graph denoted $G = (V, A)$, where V is the set of nodes and A the set of arcs. The set V consists of three disjoint subsets denoted V^{du}, V^{ac}, and V^{ap} and corresponding to different types of nodes, namely the distributed unities, aggregation/core servers, and application nodes, respectively. Each node $u \in V$ is characterized by a set of available capacities

1. The content of this chapter was accepted for publication on Transactions on Network and Service Management: W. d. S. Coelho, A. Benhamiche, N. Perrot and S. Secci, "Function splitting, isolation, and placement trade-offs in network slicing", November 2021, Preprint.

Table 5.1 – Main notation: sets and parameters

Set	Description
V	Set of all nodes.
V^{du}	Set of all distributed units.
V^{ac}	Set of all aggregation/core nodes.
V^{ap}	Set of all applications server nodes.
A	Set of all arcs.
F	Set of all NFS types.
F^d	Set of all data-plane NFS types.
F^c	Set of all control-plane NFS types.
S	Set of all network slice requests.
$F(s)$	Set of all CP NFS pairs that must be connected in slice s.
$G(s)$	Set of all pairs of NFSs from different type sets that must be connected to each other in slice s.
$K(s)$	Set of all demands of slice request s.
$O(s)$	Set of origin nodes of all traffic demand from slice s.
N	Set of all NFs.
C	set of resource types available on physical nodes
Parameter	**Description**
c_u^c	amount of available resource c on node u.
μ_u^c	cost of one unit of resource c provided by node u.
b_a	bandwidth value on arc a.
d_a	delay value on arc a.
c_f^c	amount of resource c required by NFS f.
$cap(f)$	traffic processing capacity of NFS f.
b_{fg}	total amount of traffic generated between NFSs f and g by an UE.
b_f	expected data rate of NFS f given one UE.
d_{fg}	the maximum accepted delay between NFSs f and g.
λ_f	compression coefficient of NFS f.
α_f^s	equals to 1 if a NFS type f must be present in slice s; 0 otherwise.
q_{fg}^{st}	holds 1 if slice s admits sharing a NFS of type f with a NFS of type g of slice t; 0 otherwise.
q^{st}	equals to 1 if slice s admits sharing physical node with slice t; 0 otherwise.
η_s	expected number of UEs connected to slice s
d_s	maximum accepted delay on data plane of slice s.
o_k	origin node of demand k
t_k	target node of demand k
b_k	expected volume of data between sent by origin node of demand k.

denoted by $C = \{c_u^1, \ldots, c_u^c\}$, corresponding to the types of physical resources, on any node type, and a cost per unit of resource usage $c \in C$, noted $\mu_u \geqslant 0$. Each arc $a = (u, v) \in A$ represents a physical link connecting nodes u and $v \in V$, and is characterized by a bandwidth capacity b_a and a latency value d_a.

5.1.2 Network function services

Let N denote the set of Network Functions (NF). Each NF $n \in N$ is composed of one or several Network Function Services (NFS). The set of all NFS is noted F. We suppose that F is composed of a sub-set F^c of control-plane NFSs, an ordered sub-set F^d of data-plane NFSs and an auxiliary dummy function f_0, so that $F = F^c \cup F^d \cup \{f_0\}$. Every NFS $f \in F$ requires a set of resources $\{c_f^1, \ldots, c_f^c\}$, has a traffic processing capacity denoted $cap(f)$ and delivers an expected data rate b_f. Moreover, we let $b_{fg} \geqslant 0$ denote the total amount of traffic generated by two communicating NFSs f and g

and by d_{fg} the maximum delay threshold between NFSs f and g. For each $f \in F^d$, λ_f denotes a compression coefficient applied on the data-plane traffic of NFS f. We suppose that $\lambda_{f_0} = 1$ and all the aforementioned parameters are supposed to be equal to 0 for the dummy NFS f_0.

5.1.3 Network slice requests

We denote by S the set of network slice requests. Every request $s \in S$ is associated with a binary parameter α_f^s that takes 1 (resp. 0) if at least one NFS of type $f \in F$ is (resp. is not) required in the associated slice. Let $F(s) = \{(f, g) : (f \in F^c) \wedge (g \in F^c)\}$ be the set of NFS types that must be connected. Additionally, we denote by $G(s) = \{(f, g) : (f \in F^c) \oplus (g \in F^c)\}$ the set of NFS pairs from different sub-sets of NFS types that must be connected. Hence, the control-plane required by a slice s is given by $F(s) \cup G(s)$. We denote by q_{fg}^{st} the binary parameter that takes value 1 if two NFS $f, g \in F$ respectively required by slice $s, t \in S$ can be packed together in the same NF, and 0 otherwise, representing then the so-called *virtual layer isolation*. Similarly, the *physical layer isolation* requirement is expressed by the binary parameter q^{st} that takes value 1 if slice requests $s, t \in S$ can share a common physical node, 0 otherwise. Each slice request $s \in S$ is associated with a set $K(s)$ of traffic demands to be routed on the physical layer. Each demand $k \in K(s)$ is defined by a pair of origin-destination nodes (o_k, t_k), an initial data rate b_k sent by o_k to t_k and a maximum end-to-end latency value d_s, similar for all traffic demands in $K(s)$. Finally, the expected number of users connected to slice s is denoted by η_s.

5.1.4 Problem statement

We define the Network Slice Design Problem as follows.

Definition 5.1.1. *Given a directed graph G representing the physical network, a set of slice requests S, a set of traffic demands $K(s)$ associated with each request $s \in S$, and a set F of NFS types, the NSDP consists in determining the number of NFSs to install for each $s \in S$, the size of NF hosting them as well as in deciding whether they are to be installed centrally or distributed (e.i., selecting the functional splitting), so that (i) $K(s)$ demands can be controlled and routed in G using these NFs; (ii) the NFSs installed on G can be packed into the NFs while satisfying both isolation and capacity constraints; and (iii) a path in G is associated with each pair of installed NFs that must be connected. The objective is to design each network slice and embed them into the physical network G while minimizing the*

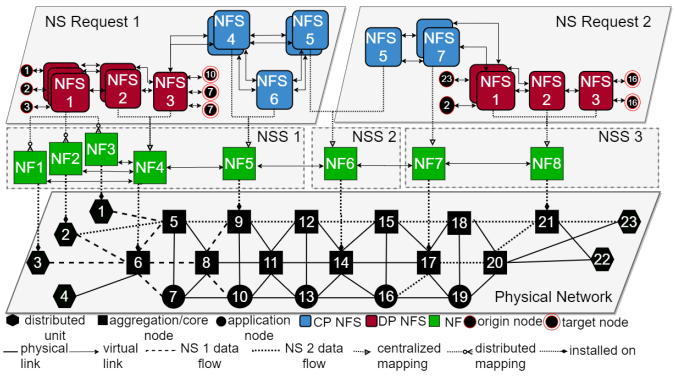


Figure 5.1 – Example of a solution for an NSDP instance.

total cost of deploying the network slice requests and ensuring all technical constraints imposed by both physical and virtual layers.

Fig. 5.1 shows an example of how to design different slices and embedding them into a physical network. In this example, we consider 2 slices, 5 traffic demands (e.g., from NS 2: distributed unit 23 $\rightarrow$ application node 16), 7 NFS types (3 for data-plane and 4 for control-plane), 8 NFs, and a physical network with 23 nodes (6 DUs, 12 aggregation/core nodes representing CUs, and 5 application nodes). Note that, for each slice, several copies of the same NFS type could be required (e.g. NFS 2). In the illustrated solution, copies of NFS 1 from NS 1 are installed distributed, at each of its origin nodes, while all other NFSs are centralized. Furthermore, copies of NFS 5 are packed into NF6 and shared by both network slices. Finally, the traffic flow from each slice request is routed through the physical network: regarding the traffic demand (distributed unit 3 $\rightarrow$ application node 7) of slice 1, its virtual DP flow encompassing the virtual link (NF1, NF4) is routed through the physical path $(3, 6, 7)$. In our previous work [9], we reported numerical results showing that flexible splitting appears as a key factor to deal with heterogeneous requirements to deploy distinct CSs, leading to considerable network slice cost decrease.

5.2 Related works

Scalability is a crucial point in dynamic environments, such as mobile networks. Authors in [84] propose an algorithm based on Control Theory in order to balance the load on instances of a specific core-based NF, called Access Management Function (AMF). Their algorithm scales out or in the AMF

instance depending on the network load in order to save both virtual and physical resources. In the same context, authors in [102] propose another solution to scale dynamically the 5G NFs; the proposed approach prevents the latency and avoids overloading the core network. Authors in [103] propose an online algorithm to minimize the cost for provisioning NF instances while minimizing the congestion in a data-center network. Authors in [105, 104] propose different proactive approaches in order to estimate the upcoming traffic and adjust NF deployment *a priori*. While [105] combine an online learning method with a multi-period online optimization algorithm, authors in [104] aim to minimize the error in predicting the service chain demands for new instance assignment and service chain rerouting. Moreover, authors in [99] address an NF scaling and sharing problem in order to minimize the redundancy throughout the virtual networks; they propose FlexShare, a near-optimal NF-sharing algorithm capable of ensuring priority and NF sharing decisions in polynomial time. Authors in [100] propose a mathematical formulation and a heuristic based on a goodness function in order to address large-sized network instances; they show that sharing NFs among network slices can use up to 30% less bandwidth and 45% fewer NF instances, compared to dedicated-NF approaches. It is important to mention that, differently from these works, our model also encompasses the control-plane and data-plane separation and considers different functional split options for designing NSs.

On top of mapping NFSs into network functions, we should provide a physical path connecting each pair of NFs that must be connected. In this context, each virtual link between two or more NFs has specific rules that must be ensured on the physical layer, such as ordering constraints, minimum bandwidth, and maximum end-to-end delay values. The objective might be to optimize the length of physical paths carrying NSs' flow while respecting the imposed technical constraints. These restrictions are the combination of technical constraints imposed by each network slice served by the given virtual structure. This sub-problem can be seen as a variant of the well-known multi-commodity flow problem [106] with additional latency and ordering constraints. Authors from [107, 108] address different aspects of this problem and propose mathematical models applied to 5G networks. In [107], the authors propose a framework that exploits the traffic information and topology of both backhaul and core networks for 5G systems; they propose a linear programming relaxation method and a heuristic method in order to better manage network load balancing, achieving close-optimal solutions with low computational complexity. Authors in [108], in turn, aim at integrating backhaul and fronthaul traffic over the same transport layer; a routing optimization framework is proposed, taking into account delay

and path constraints, as well as a heuristic to reduce the computational complexity and apply it to production-level networks.

Optimization approaches related to network slicing problems mostly consider them either as Virtual Network Embedding (VNE) [109] or Function Placement and Routing (FPR) [110] problems. The former problem can be defined as follows: given a set of r requested virtual network represented by graphs $G_v = (V_v, A_v)$, $v \in \{1, \ldots, r\}$ and a physical network represented by graph $G_p = (V_p, A_p)$, the aforementioned approaches seek to embed the graphs G_v into G_p. Hence, each requested node $u \in V_v$, $v \in \{1, \ldots, r\}$ is mapped to a physical node in V, each requested arc $a \in A_v$ is mapped to a physical path on G_p, and all technical constraints are respected. In the case of the FPR problem, each virtual network request is given by a directed acyclic graph to represent ordering constraints on mapping virtual nodes through physical paths connecting different pairs of source-destination nodes on the physical network. For instance, Esteves et al [111] propose an FPR-based ILP formulation to design network function placement under slice-mimicking demands while considering the users' geographic location to guarantee the acceptable end-to-end latency on the data-plane flow; the same authors propose in [112] an online heuristic to address the computational complexity of the studied problem. In the same context, Fendt et al [113] present a MILP taking into consideration the network function chaining and path splitting, which is based on an FPR formation. The proposed model also considers an embedding with minimum latency for the virtual links of the NF chains related to the slices. Liu et al [114], in turn, consider that the quantity, the types, and the locations of the NFs related to the slice are determined by the requirements and distribution of users. Due to the complexity of the proposed model, they also present a VNE-based heuristic to address the related problem in large-scale networks.

Even though several works partially cover the network slice design problem [97, 98] and related sub-problems, such as functional split mode selection [63, 64, 56], network slicing with VNF sharing [99, 100, 101], and network slicing with VNF scaling [84, 102, 103, 104, 105], no attention has been given to address jointly all the aforementioned aspects in order to design network slices and understand the impact of mapping, sharing and split policies.

5.3 Mathematical programming formulation

We now present the network slice provisioning as an optimization problem including novel mapping and provisioning requirements rising with new radio and core function placement policies. In this section, we propose a mixed-integer linear programming formulation for the NSDP. The problem variables are defined as follows:

• $z_f^s \in \{0,1\}$ that takes value 1 if NFS f is centralized, 0 otherwise,

• $x_{nu}^{sf} \in \{0,1\}$ that takes value 1 if NFS f serving slice s is packed into NF n, installed on node u, 0 otherwise,

• $w_{nu}^{sf} \in \mathbb{R}_+$ the ratio between the quantity of traffic processed by NFS f, packed into NF n and installed on node u to serve slice s, over its capacity $cap(f)$,

• $y_{nu}^{f} \in \mathbb{Z}$ the total number of NFSs f packed into NF n, installed on node u,

• $\gamma_{fg}^{ka} \in \{0,1\}$ that takes value 1 if arc a is used to route the traffic between NFS f and NFS g for demand k, 0 otherwise.

The constraints are categorized into several blocks and defined as follows :

Split Selection: The *split selection constraints* are to impose that if NFS $f \in F^d$ is centralized, then the remaining NFSs of the ordered set F^d should also be centralized, thus defining a splitting policy attached to a slice :

$$z_f^s \leq z_{f+1}^s, \qquad \forall s \in S, \forall f \in F^d \backslash \{f_{|F^d|}\}. \tag{5.1}$$

Network Function Services Dimensioning and Packing: Equalities (5.2) are to define variables w directly as a function of x variables, while inequalities (5.3) force to install enough NFS to process all the traffic. Inequalities (5.4) and (5.5) are the isolation constraints, respectively on NFs and on physical nodes. Finally, constraints (5.6) ensure that NF n is assigned to at most one physical node

of V.

$$cap(f)w_{nu}^{sf} = \begin{cases} \sum\limits_{k \in K(s)|u=o_k} \lambda_{f-1} b^k x_{nu}^{sf} & \text{if } f \in F^d, u \in V^{du} \\ \eta_s b_f x_{nu}^{sf} & \text{if } f \in F^c; \\ \sum\limits_{k \in K(s)} \lambda_{f-1} b_k x_{nu}^{sf} & \text{if } f \in F^d, u \in V^{ac}. \end{cases} \quad \forall s \in S, \forall f \in F, \forall n \in N, \forall u \in V \tag{5.2}$$

$$\sum_{s \in S} w_{nu}^{sf} \leq y_{nu}^{f} \qquad \forall n \in N, \forall v \in V, \forall f \in F \tag{5.3}$$

$$x_{nu}^{sf} + x_{nu}^{tg} \leq 1 + q_{fg}^{st} q_{gf}^{ts} \qquad \forall s,t \in S, u \in V, n \in N, f,g \in F \tag{5.4}$$

$$\sum_{n \in N} x_{nu}^{sf} + \sum_{m \in N} x_{mu}^{tg} \leq 1 + q^{st} q^{ts} \qquad \forall s,t \in S, u \in V^{na}, f,g \in F \tag{5.5}$$

$$x_{nu}^{sf} + x_{nv}^{tg} \leq 1 \qquad \forall s,t \in S, f,g \in F, n \in N, u,v \in V : v \neq u \tag{5.6}$$

Let us explain in detail the inequalities (5.3) with some examples. Suppose that NFSs of type f from s and t cannot be packed together ($\forall n \in N, x_{nu}^{sf} \oplus x_{nu}^{tf}$). Hence, all copies of f installed on node u and serving s are not shared with t. In this way, if (5.2) set w_{mu}^{sf} to 4.60 and w_{nu}^{tf} to 1.25, for example, we must install at least seven ($\lceil 4.60 \rceil + \lceil 1.25 \rceil$) NFSs of type f on the node u using two different NFs. Now, let s and t be two slices with no isolation constraints and using the same NF for a given NFS f ($x_{nu}^{sf} \wedge x_{nu}^{tf}$). Suppose that (5.2) have set w_{bu}^{sf} and w_{nu}^{tf} equal to 4.60 and 1.25, respectively. Since both s and t accept NFS sharing with each other ($q_{ff}^{st} \wedge \mathrm{q}_{ff}^{ts}$), we need to install only six ($\lceil 4.60 + 1.25 \rceil$) NFSs of type f on node u instead of seven of them. Using this approach on residual capacities, this saving can be even greater if we have a bigger sub-set of slices having $q_{fg}^{st} = 1$ for a given tuple (s,t,f,g).

Network Function Services Placement: For each slice request s, equalities (5.7) ensure that a distributed NFS f should be installed on every origin node of $K(s)$; whereas equalities (5.8) ensure that each centralized NFS f serving s should be installed in a node of V^{ac}.

$$\sum_{n \in N} x_{nu}^{sf} = \begin{cases} 1 - z_f^s & , \text{ if } f \in F^d, u = o_k; \\ 0 & , \text{ otherwise .} \end{cases} \qquad \forall k \in k(s) | s \in S, \forall f \in F, u \in V^{du} \tag{5.7}$$

$$\sum_{n \in N} \sum_{u \in V^{ac}} x_{nu}^{sf} = \begin{cases} z_f^s & , \text{ if } f \in F^d; \\ \alpha_f^s & , \text{ if } f \in F^c. \end{cases} \qquad \forall s \in S, \forall f \in F \tag{5.8}$$

Traffic routing: The constraints (5.9) are the flow conservation constraints, for each slice request $s \in S$, each demand $k \in K(s)$ and each pair of NFSs in F. They allow to associate a path in G for each traffic demand k between its origin node o_k and the first NFS $f = 1$ from the ordered set F^d and serving k, and between the last NFS $f = |F^d|$ from F^d and the destination node t_k of k.

$$\sum_{a\in\delta^+(u)} \gamma^{ka}_{fg} - \sum_{a\in\delta^-(u)} \gamma^{ka}_{fg} =$$

$$\begin{cases} z^s_f - 1 & \text{if } (f,g)\in G(s), f\in F^c, u = o_k, \\ 1 - z^s_f & \text{if }, u = o_k, ((f,g)\in G(s), f\in F^d) \oplus (f = f_{|F^d|}, g = f_0); \\ -\sum\limits_{n\in N} x^{sg}_{nu} & \text{if } u\in V\backslash V^{du}, f = f_0, g = f_1 | g\in F^d \\ z^s_g & \text{if } u = o_k, f = f_0, g = f_1 | g\in F^d \\ -1 & \text{if } u = t_k, f = f_{|F^d|}, g = f_0 \\ \sum\limits_{n\in N} x^{sf}_{nu} & \text{if } u\in V\backslash V^{du}, f = f_{|F^d|}, g = f_0 \\ z^s_g - z^s_f & \text{if } u = o_k, \forall f,g\in F^d | g = f+1 \\ \sum\limits_{n\in N} x^{sf}_{nu} - \sum\limits_{m\in N} x^{sg}_{mu} & \text{otherwise.} \end{cases}$$

$$\forall k\in K(s) : s\in S, \forall f,g\in F, \forall u\in V \qquad (5.9)$$

Latency: Inequalities (5.10) ensure that each demand $k\in K(s)$ is routed along a path that respects the end-to-end latency value requested for slice $s\in S$ while inequalities (5.11) ensure that the maximum latency value between any pair of NFSs is also respected.

$$\sum_{a\in A} d_a(\gamma^{ka}_{f_{|F^d|}f_0} + \sum_{f\in\{f_0\}\cup F^d\backslash\{f_{|F^d|}\}} \gamma^{ka}_{ff+1}) \le d_s \qquad \forall k\in K(s) : s\in S \qquad (5.10)$$

$$\sum_{a\in A} d_a\gamma^{ka}_{fg} \le d_{fg} \qquad \forall k\in K(s) : s\in S, \forall f,g\in F \qquad (5.11)$$

Physical Capacity: Inequalities (5.12) are the capacity constraints over the arcs of A. Note that the flow using an arc $a\in A$ is composed of two types of traffic for each $s\in S$, namely the part of traffic generated by inter-NFS communication and the part of traffic generated by the demands of $K(s)$, which is submitted to the compression coefficients λ. Inequalities (5.13) express the capacity constraints in terms of NFs that can be installed on each physical node $u\in V$.

$$\sum_{s\in S}\sum_{k\in K(s)} b^k(\lambda_{f_{|F^d|}}\gamma^{ka}_{f_{|F^d|}f_0} + \sum_{f\in\{f_0\}\cup F^d\backslash\{f_{|F^d|}\}} \lambda_f\gamma^{ka}_{ff+1})]$$
$$+\sum_{s\in S}\eta_s(\sum_{(f,g)\in F(s)} b_{fg}\gamma^{1a}_{fg} + \sum_{(f,g)\in G(s)}\sum_{k\in K(s)} \frac{b_{fg}\gamma^{ka}_{fg}}{|K(s)|}) \le b_a \qquad \forall a\in A \qquad (5.12)$$

$$\sum_{n\in N}\sum_{f\in F} c^c_f y^f_{nu} \le c^c_u \qquad \forall u\in V, \forall c\in C \qquad (5.13)$$

Being Ω the scaling coefficient related to link utilization, the NSDP is then equivalent to the following formulation:

$$min \sum_{f\in F}\sum_{n\in N}\sum_{u\in V} y^f_{nu} + \Omega\sum_{a\in A}\sum_{s\in S}\sum_{k\in K(s)}\sum_{f,g\in F} \gamma^{ka}_{fg} \qquad (5.14)$$

$$s.t. \quad (5.1) - (5.13)$$

$$y^f_{nu} \geq 0 \qquad \in \mathbb{Z} \qquad \forall f \in F, \forall n \in, \forall u \in V \tag{5.15}$$

$$x^{sf}_{nu} \qquad \in \{0,1\} \qquad \forall s \in S, \forall f \in F, \forall n \in, \forall u \in V \tag{5.16}$$

$$z^s_f, \qquad \in \{0,1\} \qquad \forall s \in S, \forall f \in F \tag{5.17}$$

$$\gamma^{ka}_{fg} \qquad \in \{0,1\} \qquad \forall k \in K(s) : s \in S, \forall f, g \in F \tag{5.18}$$

$$w^s_{uf} \geq 0 \qquad \in \mathbb{R} \qquad \forall s \in S, \forall f \in F, \forall n \in, \forall u \in V \tag{5.19}$$

While the first term in (7.9) is related to the number of installed functions, the second term in (7.9) multiplied by Ω is inserted to mechanically avoid loops, with a negligible qualitative impact on the network solution. An alternative way to the second term would be to add loop avoidance constraints, however increasing the complexity. Network designers may want to tune the factor Ω to drive toward the desired outcome (e.g., to emphasize the number of NFSs over the number of links).

5.4 Complexity

In order to prove the complexity of the NSDP, we first define the Virtual Network Embedding problem, which is used in our reduction framework.

Virtual Network Embedding problem: Let $G_p = (V_p, A_p)$ be a directed graph representing the physical network (also known as substrate network), where V_p is the set of physical nodes and A_p is the set of physical links. While $c^c_p(u)$ represents the available resource $c \in C$ on the physical node $u \in V_p$, $b_p(a)$ is the available bandwidth on arc $a \in A_p$. Also, each arc in A_p has the latency $d_p(a)$ expressing the time needed by a fow to traverse a. Finally, the request graph is similarly defined as a directed graph $G_r(V_r, A_r)$ with minimum capacities $c^c_r(u)$ and $b_r(a)$ required respectively by virtual nodes and arcs from G_r; each arc in A_r also has the maximum latency $d_r(a)$ that must be respect in the embedding process.

In the VNEP, the requested graph must be mapped to the physical graph: each requested node $u \in V_r$ is mapped to a physical node in V_p , and a requested arc is mapped to a physical path on G_p. A valid mapping must respect both required and available resources and latency constraints. As proven by [115], the VNEP is $\mathcal{NP}$-complete even with no latency and resource constraint related to

the physical network (i.e., physical nodes and links have infinite capacities and no latency) and cannot be approximated under any objective unless $\mathcal{P} = \mathcal{NP}$ holds.

Theorem 5.4.1. *The Network Slice Design problem is $\mathcal{NP}$-complete even with only one slice request.*

Proof. We first prove the $\mathcal{NP}$-hardness of the NSDP. For this purpose, our framework reduction relies on the VNEP. Given a VNEP instance with a request graph G_r, we construct a slice request s for each sub-graph in G_r as following. For each requested node $u \in V_r$, a control-plane NFS is created in F^c with the same required capacity c^c_u and, for each $f \in F^c$, the related α^s_f is set to 1. Similarly, for each arc in G_r, there is a tuple in $F(s)$ imposing the connection between the related NFSs f and g from F^c with the same acceptable minimum bandwidth and maximum latency values. In addition, a set F^d is constructed with dummy functions with no capacity required to be installed (i.e., $c^c_{f_0} = 0,\ \forall f \in F^d, \forall c \in C$). Additionally, any function in F has infinite processing capacity (i.e., $cap(f) = +\infty$). For any tuple $(s, t, f, g) : \{s, t \in S, f, g \in F | s \neq t\}$, the related isolation parameter q^{st}_{fg} is set to 0. In other words, no NFS sharing is allowed between any pair of slice requests. Moreover, for each $s \in S$, dummy traffic demands with no data flow are created (i.e., $b_k = 0,\ \forall k \in K(s) | s \in S$); origin and destination nodes o_k and t_k are randomly chosen among the physical nodes in V^{du} and V^{app}, respectively, for any traffic demand k. Note that F^d and $K(s)$ sets can be of any size, even empty. Finally, the graph G representing the physical network remains the same. This reduction is done in polynomial time with complexity $O(|A_r|)$.

Clearly, if there exists a feasible solution to the NSDP that respects the capacity and latency constraints (if there exists any), then this solution is also feasible for the VNEP. Hence, any algorithm applied to solve the NSDP can also be used to decide the VNEP, showing the $\mathcal{NP}$-hardness of the NSDP.

To conclude the $\mathcal{NP}$-completeness proof, we now show that NSDP is also in $\mathcal{NP}$. For this end, our verifier-based algorithm relies on the proposed compact formulation (5.1)-(5.19) without the objective function (7.9). Giving an NSDP instance $\mathcal{I}$, a certificate $\mathcal{X}$ (i.e., a solution to $\mathcal{I}$) can therefore be constructed in polynomial size in $|S|$, $|F|$, $|N|$ (i.e., number of variables). In other words, $\mathcal{X}$ is a vector with a fixed value to each variable of the linear system. Our verifier $\mathcal{V}$ is also constructed in polynomial size in $|S|$, $|F|$, $|N|$ and $|V|$ (i.e., number of constraints). To decide whether $\mathcal{X}$ is a feasible solution to $\mathcal{I}$, we apply $\mathcal{V}$ in order to verify if there exists at least one violated constraint. It is easy to

see that $\mathcal{V}$ returns *true* if and only if $\mathcal{X}$ is a feasible solution to $\mathcal{I}$ (i.e., no constraint is violated). Since verifying a system of equations is done in a polynomial time, the NSDP is also in $\mathcal{NP}$, and hence in $\mathcal{NP}$-complete. □

It is important to note that NF dimensioning and NFS sharing sub-problems are partially solved by setting all related isolation parameters q_{fg}^{st} to zero; this implies that, on the control-plane, there exists only one feasible solution for each sub-problem; on the data-plane, these sub-problems depend on the select split setting chosen for each slice request. Also, even being fully solved, the data-plane split selection sub-problem has no impact on the reduction framework since no physical capacity is required by the related dummy flows. Note that this sub-problem can be seen as a Service Function Chaining problem [116], which is also a difficult problem. Clearly, an NSDP instance that has different values for these aforementioned parameters increases the related sub-problems' feasible solution space and therefore makes the NSDP even harder to be solved in optimization scenarios.

5.5 Variants and extensions for the NSDP

In the following, we present a few relevant variants of the Network Slice Design Problem.

5.5.1 NDSP with intra-slice flexible splitting

With this variant of the problem, different split settings can be selected within the same slice. In other words, flexible splitting is applied independently to each DU related to a given slice. For this purpose, we apply a pre-processing to transform each traffic demand into a slice request[2]. Hence, any NS request is now composed of only one traffic demand (i.e., representing a unique traffic demand of the initial NS request). In order to impose a shared control-plane to all DU related to the same initial NS (i.e., before the pre-processing), we introduce β_{st}, a binary parameter generated during the pre-processing: it holds 1 if the new slices s and t come from the same initial NS request (i.e., before the decomposition); 0 otherwise. Finally, in order to reduce the management complexity, we add the new constraints (5.20): they impose that requests from the same NS must share the same control-plane NFSs. Note that the single requests from the same original slice can have their own data-plane. We

2. Following the taxonomy presented in this work, these new post-processed requests can be seen as network slice subnets.

refer to this variant by NSDP with intra-slice flexible splitting (NSDP-ISFS).

$$\beta_{st} - 1 \leq x_{nu}^{sf} - x_{nu}^{tf} \leq 1 - \beta_{st} \qquad ,\forall s,t \in S, f \in F^c, n \in N, u \in V_{ac} \tag{5.20}$$

Note that, constraints (5.20) impose the binary variables x to have the same value (i.e., either 1 or 0) if and only if the related parameter β holds 1; otherwise, these inequalities are implicitly relaxed. After applying the described pre-processing on the initial input, the original formulation of NSDP (5.1)-(5.19) can be directly applied along with the new constraints (5.20).

5.5.2 NDSP with inter-slice split continuity

We propose this variant in order to represent the scenarios with strict split setting constraints on each DU. In fact, imposing the same split selection for any traffic demand traversing a given DU might be necessary to reduce the management complexity. Complementary to Ineq. (5.20), we add the new constraints (5.21), where ρ_{st} is a binary parameter generated during the pre-processing; it holds 1 if the new slices s and t have the same origin DU node as their traffic demands; 0 otherwise.

$$\rho_{st} - 1 \leq z_f^s - z_f^t \leq 1 - \rho_{st} \qquad ,\forall s,t \in S, \forall f \in F^d \tag{5.21}$$

Note that these inequalities can only be applied to instances whose slice requests have only one traffic demand (i.e., after pre-processing). We refer to this variant by NSDP with inter-slice split continuity (NSDP-ISSC).

5.5.3 NSDP with optimized link load

In order to minimize the traffic volume throughout the network, we introduce U, a continuous variable that represents the maximal load among physical links. We then replace constraints (5.12) by Ineq. (5.22) and (5.23) and add the new constraints (5.24) in order to impose upper bounds to y variables; these inequalities are important to this NSDP variant since we no longer have the related component within the new objective function (5.25).

$$\sum_{s \in S} \sum_{k \in K(s)} b^k (\lambda_{f_{|F^d|}} \gamma^{ka}_{f_{|F^d|} f_0} + \sum_{f \in \{f_0\} \cup F^d \setminus \{f_{|F^d|}\}} \lambda_f \gamma^{ka}_{f f+1})] +$$

$$\sum_{s \in S} \eta_s \Big(\sum_{(f,g) \in F(s)} b_{fg} \gamma_{fg}^{ksa} + \sum_{(f,g) \in G(s)} \sum_{k \in K(s)} \frac{b_{fg} \gamma_{fg}^{ka}}{|K(s)|} \Big) \leq b_a U \qquad , \forall a \in A \tag{5.22}$$

$$0 \leq U \leq 1 \tag{5.23}$$

$$y_{nu}^{f} < 1 + \sum_{s \in S} w_{nu}^{sf} \qquad , \forall n \in N, \forall v \in V, \forall f \in F \tag{5.24}$$

The new objective function is then formulated as following:

$$min\ U \tag{5.25}$$

This formulation can be applied to any NSDP variant and a similar model can be generated in order to minimize the maximal load on physical nodes.

5.6 Sensibility analyses

We now propose an open-access framework based on the proposed MILP formulations, which encompass flexible functional splitting, with possibly different splitting for different slices and slice subnets, while taking into account different network sharing policies from 5G specifications. We also consider novel mapping and continuity constraints specific to the 5G architectures and beyond. We show by numerical simulations the impact of taking into full and partial consideration these peculiar novel technical constraints. To this purpose, we first detail the simulation setting and then expose the results.

5.6.1 Simulation setup

Let us detail the simulation settings.

5.6.1.1 Physical topologies

We simulated different physical networks with different features. Inspired by common access networks structure, we first propose a specific topology called *Mandala* (Fig. 5.2a) with the following structure: given n DU nodes, we have $n/4$ aggregation nodes, $n/4$ core nodes, and $n/8$ application nodes. Note that n must be equal or multiple of 8. Each DU node is connected to two aggregation nodes, which, in turn, are connected to two inner-level core nodes. Each core node is additionally connected to two application nodes, where demands are served. Finally, given two different nodes u

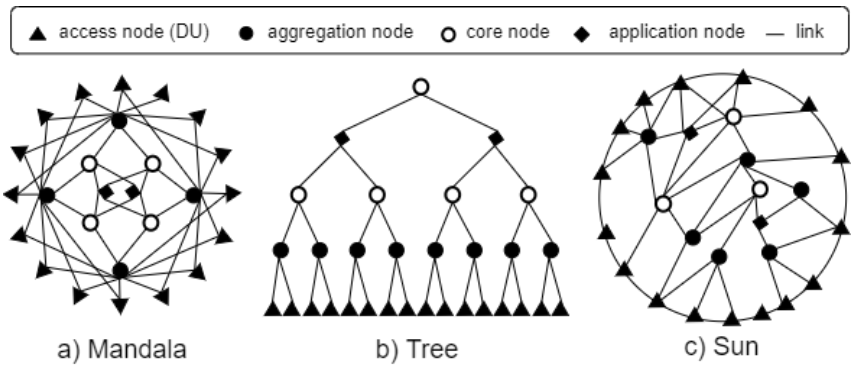


Figure 5.2 – Physical network structures: examples with 16 DUs.

and v, there exists one arc (v, u) for each arc (u, v). Fig. 5.2a shows this topology where n is equal to 16. For sake of clarity, each pair of arcs between two nodes is represented by an edge.

In our simulation, while application nodes have no capacity constraint (they are considered as sink nodes), each one of DU, aggregation, and core nodes provides 30 servers, each of which with 16 CPUs; this capacity corresponds to 12.5% of the global generated CPU computing demand (i.e., with no function sharing) and enables to test all split settings and sharing policies. In addition, fronthaul links (i.e., between DUs and aggregation nodes), backhaul links (i.e., between aggregation and core nodes), and core links (i.e., between core and application nodes) have link capacities b_a set to respectively 100%, 200% and 300% of the maximum flow sent by a single DU at the split setting with the highest bitrate. Finally, to simulate a small region, the latency d_a in each arc randomly takes a value between: 50μs and 100μs for fronthaul links, 200μs and 300μs for backhaul links, and 400μs and 600μs for core links[3].

We also run our tests on two different physical topologies: one binary tree-based structure (hereinafter referred to as *Tree*; Fig. 5.2b) with 31 nodes and 60 arcs, and *Sun* from SNDlib [118] composed of 27 nodes and 102 arcs (Fig. 5.2c). We mapped the 16 DUs to all 16 leaves and the nodes composing the external ring path in the former and latter structures, respectively; aggregation, core, and application nodes were randomly mapped in both topologies. While the capacities on physical nodes follow the same parameter values in Mandala, the bandwidth on links from the Tree structure was set to 500% of the maximum flow sent by a single DU at the split setting with the highest bitrate; the latency is between 50μs and 100μs. For the Sun topology, these values were randomly chosen between

3. Note that the end-to-end latency along the shortest path between any DU and application node is at most 1ms. This value is commonly used as a threshold to strict latency constrained 5G services [117].

Table 5.2 – Simulated slice demand setting.

Slice	Service required	Aditional CPNFSs	Max E2E latency d_s	UE data rate	UE per DU
1	Broadband access in dense areas	NFS10, NFS11	10ms	300Mbps	600
2	Ultra-low cost broadband	-	10ms	10Mbps	600
3	Real-time communication	NFS11, NFS12, NFS13	1ms	25Mbps	180
4	Video broadcast	NFS10, NFS11	100ms	200Mbps	60

50μs and 600μs for the latency whereas the bandwidth values were set between 100% and 300% of the maximum flow sent by a single DU.

5.6.1.2 Virtual layer

To scale with the complexity of the formulation while stressing the impact of functional splitting on the placement of NFSs, we set F^d with five data-plane NFS types: NFS1 represents functions of the MAC bloc; NFS2 represents functions of the RLC block; NFS3 represents functions from PDCP block; NFS4 represents functions from RRC block; NFS5 represents DP functions from the core network[4]. In addition, there are four mandatory control-plane NFS types (labeled NFS6..NFS9) and other four optional CP NFS types (labeled NFS10..NFS13; examples of mandatory and optional 5G core NFs are presented in [1]). Each NFS has a processing capacity $cap(f)$ set to 100% of the average volume sent by all DUs. Furthermore, the resource c_f required to install each copy of them is set to roughly 5% of the average capacity available on physical nodes. Also, the traffic generated from or to any CP NFS was set to 1 kbps per UE.

According to the 4G functional split levels reported in Table 3.2 and considering the uplink direction, we set similar compression coefficients λ_f related to initial volume sent by a traffic demand: 65% for NFS1 and 40% for the other DP NFSs. Additionally, the acceptable latency d_{fg} between two DP NFSs from F^d also follows those in Table 3.2, taking the upper bound when an interval is proposed. Finally, the latency d_{fg} involving any CP NFS is set not to exceed 500μs; this value corresponds to 5% of the total CP latency proposed by 3GPP [6].

5.6.1.3 Slice requests

We tested instances with four NS requests, each with four traffic demands with random origin-destination pairs; for each $k \in K(s)$, origin o_k is a DU while destination t_k is an application node as

4. Since RF and PHY blocs have synchronous network functionalities that pose extremely strict latency requirements, we assume they are PNFs integrated to each DU. Hence, they are not considered in our virtual DP chains.

previously discussed. Additionally, all network slices must contain all data-plane NFSs, four mandatory control-plane NFSs, and a different set of additional NFSs that can be required (see Table 5.2). We assume that all CP NFSs are connected to each other. Furthermore, to simulate the communication between data and control planes, there exists an expected traffic volume between CP NFSs and DP ones on each related network slice; we create such traffic from CP NFS6 only (e.g., corresponding to the Access and Mobility Management Function, AMF, in 5G core [1]) to all DP NFSs (NFS1..NFS5).

To also observe the impact of different sharing policies on the number of distributed NFSs, 25% of available DUs are set to be an origin node of all NS requests; application nodes are evenly distributed as target nodes. Finally, each slice request imposes different technical constraints related to end-to-end latency d_s, demands for optional CP NFSs, and expected user experienced data rate. As depicted in Table 5.2, we applied the assumptions proposed by [117] for each aforementioned requirements. In our simulations, slice request 1 represents an eMBB application with an important traffic volume, which impacts both virtual and physical capacities. Slice request 3, in turn, represents an URLLC application, imposing a strict end-to-end latency on the data-plane, which restrains the placement possibilities of the related NFSs. The other two slice requests are intermediate regarding both aforementioned parameters; request 2 can be seen as an mMTC application. Finally, being an origin of one of some slice's traffic demands, each DU is associated with a flow rate equal to the product between the expected number of UE per DU and their related data rate in such NS.

5.6.1.4 Scenarios

Following Fig. 3.3, each scenario represents one combination of functional split setting and sharing policy applied to all slices. While different sharing policies are those previously presented (see Fig. 3.9), the split settings impose different sets of distributed and centralized DP NFSs. Table 5.3 summarizes the tested scenarios. Te proposed scenarios represent realistic areas, such as small cities and dense zones to scale with the complexity of the formulation while stressing the impact of functional splitting on the placement of network functions services. Each simulated parameter follows those proposed in related technical documents [117, 6] in order to provide scenarios that are as realistic as possible.

Table 5.3 – Scenarions: split settings and sharing policies

Split Setting	Description
Setting 1	all DP NFS are installed locally for all NS requests.
Setting 2	for each slice, only NFS5 is installed centrally.
Setting 3	for each slice, only NFS4 and NFS5 are installed centrally. It corresponds to 3GPP's split 1 in Fig. 3.3.
Setting 4	for each slice, only NFS1 and NFS2 are distributed; it corresponds to 3GPP's split 2 in Fig. 3.3.
Setting 5	for each slice, only NFS1 is installed locally. It corresponds to 3GPP's split 4 in Fig. 3.3.
Setting 6	all DP NFSs are installed centrally for all NS requests. It corresponds to 3GPP's split 6 in Fig. 3.3.
Flexible	free functional split selection for each NS request.
Policy	**Description**
Hard Isolation	NS requests do not accept sharing any NFS.
Shared DP	only DP NFSs can be shared among slices.
Shared CP	only CP NFSs can be shared among slices.
Partial DP Isol.	only NFS1, NFS2, and NFS3 can be shared.
Partial CP Isol.	only mandatory CP NFSs can be shared among NSs.
Flat Sharing	NS requests do not impose any isolation constraint.

5.6.2 Numerical results

The analyses made in the following sub-sections are related to the formulation (1)-(26) and present the numerical experiments of each variant of the problem on sub-section 3. We implemented our model in a Julia-JuMP environment using ILO CPLEX 12.8 as the linear solver. We set Ω to an enough small value (i.e., 10^{-3}) on the objective function (7.9) only to prioritize elementary paths to carry traffic demands and to emphasize the number of NFSs over the number of links in the optimization process. Finally, our tests were run on a Linux server with an Intel Xeon E5-2650 CPU and 256GB RAM. The data-set and the source code are available on [119].

5.6.2.1 Execution time

Before discussing the results related to the simulation setting as previously detailed, we present the performance of our model on different instance sizes: we varied the number of NFSs available (from 3 up to 24), the number of traffic demands per slice request (from 1 up to 16), and the physical topology size (13, 26, and 52 physical nodes with random connection and average degree equals to 10). Finally, all instances had 4 NS requests. We run 30 tests of each instance size, varying both origin and target nodes of each traffic demand. Finally, we set the maximal number of parallel threads that could be invoked by the solver to 1 and the time limit to 10 800 seconds (three hours).

As shown in Fig. 5.3, the time needed to achieve the best solution increases exponentially with the size of the instance. In particular, the number of NFSs available has the worst impact on the

model. Even with small topologies (13 nodes), the problem with 24 NFSs could not be solved within 3 hours. Due to the different levels of packing problems within the model, this time limit is also reached with the biggest topologies (52 nodes) and only 12 NFSs. The execution time also increases as the number of traffic demands increases. Due to the related routing sub-problems, the limit of three hours is also reached with any number of demands and 12 NFSs on large topologies. It is worthwhile to mention that, for those instances with 13 (resp. 52) nodes that reached the time limit and could not be solved to optimality by the solver, the average relative gap was roughly 7% (resp. 43%) with standard deviation equals to approximately 2% (resp. 8%).

5.6.2.2 Functional split and NF sharing

We now discuss the results depending on the presented network settings. We applied additional constraints to impose the desired split setting to all slice demands. Additionally, sharing policies were imposed by changing the q^{st}_{fg} parameters values used in Ineq. (5.4). All instances were generated using Mandala, Sun and Tree topologies with 16 DUs (see Fig. 5.2). Finally, we run 10 tests on each physical network varying both traffic demands' origin and destination nodes. The goal of the following numerical analysis is to assess the impact of novel mapping, splitting, and sharing policies on network design.

Fig. 5.4 reports the average number of distributed and centralized NFSs on different sharing policies and split strategies for the three aforementioned physical topologies merged together (i.e., in the same results set here). While distributed entities are only NFSs from DP, centralized ones also aggregate NFSs from CP; translucent bars show the total number of installed NFSs. Note first that the generated instances' characteristics are such that:

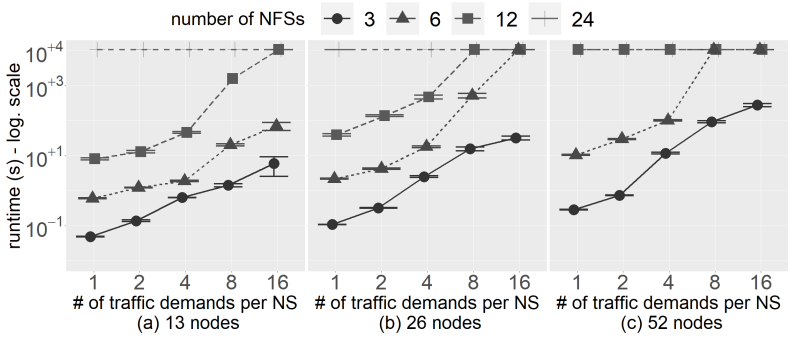


Figure 5.3 – Runtime on different NSDP instance sizes.

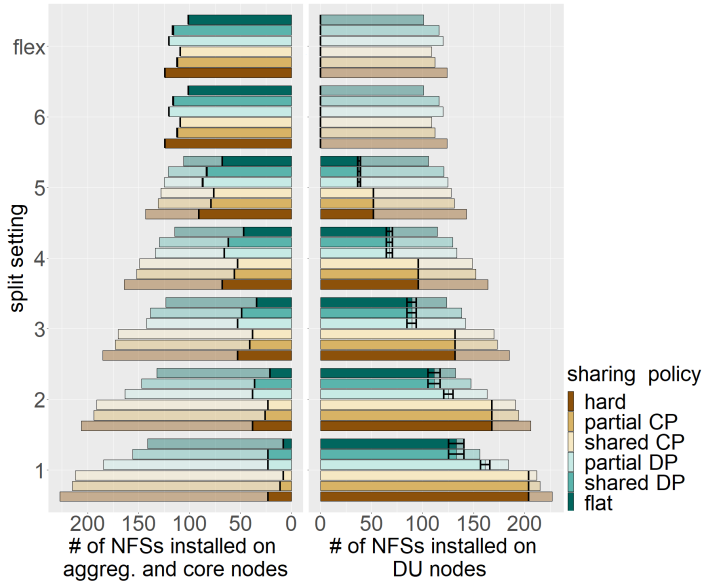


Figure 5.4 – Number of NFSs on different scenarios.

— The minimum (resp. maximum) number of NFSs required to serve all NS requests is equal to 101 (resp. 227);

— Since all NFSs are installed on all DUs related to each slice request, split setting 1 requires the largest number of NFS copies in all proposed sharing policies;

— Since each (resp. no) NFS copy is dedicated to a single NS, Hard (resp. Flat) Isolation has the greatest (resp. smallest) number of NFSs copies on all split settings, including the flexible one.

In our simulations, having isolation constraints on different sets of NFS types led to different impacts on the network slice design. Regarding the five first split settings, Shared DP and Partial DP policies provided a mean decrease (resp. increase) of 28% (resp. 42%) on the number of distributed (resp. centralized) NFSs compared to Shared CP and Partial CP; the total number of NFSs when using shared policies (i.e., CP Shared and DP Shared) was always smaller than when using partial policies (i.e., Partial CP and Partial DP). Also, flexible splitting proves to be an interesting strategy even for scenarios that have strong isolation restrictions. With roughly 56% as an overall reduction, this approach has the smallest number of NFSs in all mapping scenarios; regarding each sharing policy, the average reduction was roughly 38% (standard deviation equals to approximately 10%) compared to split setting 1. It is important to note that, since we minimize the total number of NFSs, flexible split always had the same number of NFS copies as split setting 6, which provided the greatest number

of centralized NFSs. This behavior might differ if the NS provider is interested in optimizing other parameters, such as the load on physical arcs.

In Fig. 5.5 we show that applying different sharing policies and split settings has also an important impact on the physical network. It is worth mentioning that we excluded from the computation of the average load the unused links and nodes, and for sake of readability, the standard deviations are not depicted; the observed ratios of the standard deviation to the mean were always less than 14%. First, we observe that split settings 6 and Flexible have the worst impact on link load in all sharing policies, requiring up to 100% of the capacity on the most loaded link (see Fig. 5.5e); the average load on backhaul and core (resp. fronthaul) links was equal to 52% (resp. 40%) applying Flat (resp. Shared DP) sharing policy and split setting 6 (see Figures 5.5c and 5.5d). This behavior is expected since all NFSs are installed centrally and the data volume sent by each traffic demand is completely decompressed before traversing the fronthaul links. Conversely, split setting 1 benefits from the impact of the compressed data and demands the least amount of capacity on the links in all mapping approaches, requiring at most 60% of the capacity on the link on average (see Fig. 5.5e). However, as shown in Fig. 5.5a, this split is one of the settings that require the largest number of links (between 77% and 82%) since the NFSs from CPs and DPs are further from each other. Besides, CP NFS6 must be connected to all distributed DP NFSs of the related NS.

We also note a strong impact of different scenarios on physical nodes. Since there exist at least one NFS type installed locally, the first five functional splits had the largest number of physical nodes hosting at least one NF (see Fig. 5.5f). We also observe a decrease of the average load on physical nodes (see Fig. 5.5g), in particular on DU nodes (see Fig. 5.5h), on all sharing policies. However, due to the completely decompressed data arriving in the centralized DP chain, a shift of behavior is observed when split setting 6 is applied (see Fig. 5.5g). Unlike physical links and aggregation and core nodes (see Fig. 5.5b and Fig. 5.5i, respectively), DU nodes benefit from functional splits where a greater number of NFSs is installed centrally. The average load on DU (resp. aggregation and core) nodes decreased (resp. increased) from roughly 43% (resp. 21%) applying split setting 1 along with Partial CP (resp. Shared CP) sharing policy to approximately 8% (resp. 75%) applying split setting 5 jointly with Flat sharing policy; the most loaded physical node (see Fig. 5.5j) provided 98% (resp. 43%) on average of its available resource applying split setting 6 (resp. setting 1) and Partial DP (resp. Flat) sharing policy. Note that, applying Hard, Shared CP, and Partial CP isolation policies,

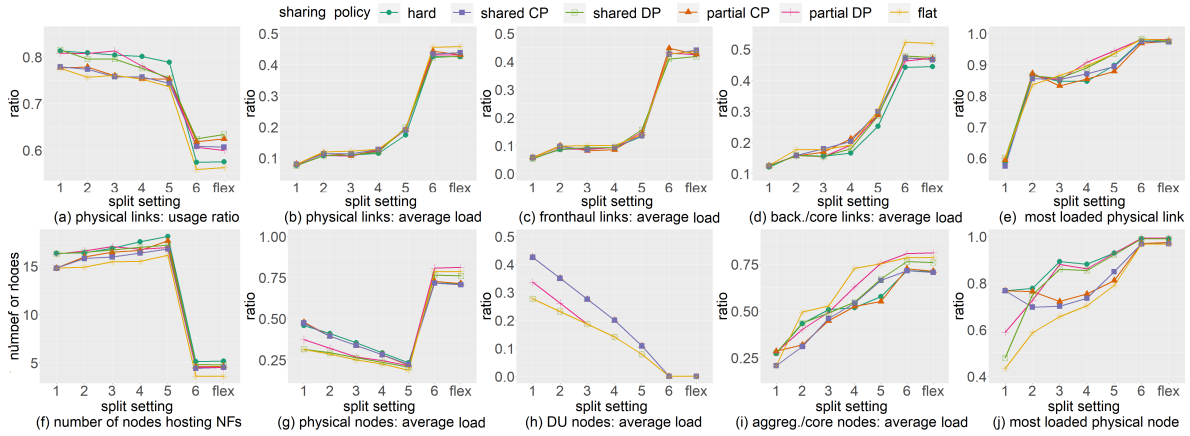


Figure 5.5 – Impact of different split settings and sharing policies on the physical network.

all distributed NFSs can serve only one slice. Hence, they will always demand the same capacity from DU nodes when the same split setting is applied (see Fig. 5.5h).

Even with a negative impact on the number of installed NFSs, mapped links, and nodes (see Figures 5.4, 5.5a, 5.5f, respectively), Hard Isolation could partially unload the physical network. In fact, due to strong isolation constraints, this sharing policy demanded less physical capacity from links (see Fig. 5.5b) and from aggregation and core nodes (see Fig. 5.5i) in some split settings. Consequently, a short physical path for each traffic demand was prioritized, leading to the use of physical nodes and links not mapped to other traffic demands. Also, let us recall that the final solutions prioritized minimizing the number of NFSs, even if this approach harms the load of the physical network; to bring the final solution closer to its economic strategy, the NS provider can simply modify the objective function (7.9) to a more suitable one. It is also worth mentioning that, in order to test feasible instances of all functional split settings, we set a low enough latency for each physical link; otherwise, some split settings (e.g. settings 5) could be impossible. Finally, since we imposed the same scenario (see Table 5.3) to all slice requests, we did not observe a significant difference in the results using distinct physical topologies. For instance, comparing the three proposed topologies, the difference in the number of physical nodes hosting an NF, the ratio of active links, and the number of NFS copies were always less than 7%, 11%, and 1%, respectively. This behavior might be different in real scenarios since NS requests are likely to impose different isolation constraints and physical networks might not have enough capacity to allow all split settings (due to the relation between the fronthaul capacity and the NFSs' compression coefficient). This, therefore, reinforces the importance of applying flexible

functional splitting while considering different sharing policies in virtual environments.

5.6.2.3 NSDP and variants

We now present the impact of the proposed variants of the presented problem on the physical network. Henceforth, we refer by *NSDP* the original formulation (5.1)-(5.19); the other two variants refer to the proposed formulations as previously discussed. Since we set Ω to 10^{-3} in (7.9), we refer by *minNFS* this objective function while *minLinkLoad* refers to (5.25); as aforesaid, this objective function is implemented along with inequalities (5.22)-(5.24). Also, the sharing policy was randomly chosen for each pair of slices while we applied the Flexible Split setting along with the same parameters related to both physical and virtual layers as previously presented in this section. Finally, we run 10 tests on each combination of objective function, variant, and physical topology, varying both origin and target nodes of each traffic demand; for NSDP-ISFS and NSDP-ISSC variants, the pre-processing described in the previous section was applied on the NSDP instances.

Figure 5.6 shows the mean and the standard deviation of each depicted parameter. Minimizing the load on physical links generally increased the number of NFSs installed throughout the network (see Figures 5.6a and 5.6b) and the number of physical nodes hosting an NF (see Fig. 5.6h); regarding all variants, the average increase in terms of both numbers of NFSs and hosting nodes was roughly 400%. This is due to the impact of the compression coefficient related to data-plane NFSs; when installed locally, they can compress the data before leaving the origin node of each traffic demand. However, as seen in Fig. 5.6j, this strategy is limited by the available resources on DU nodes, which impose to install some DP NFSs centrally whenever the related physical capacities are reached (see Fig. 5.6k and Fig. 5.6l). Moreover, since the data flow is spread over as many physical links as possible, minimizing the load on physical links also increased the end-to-end DP latency (i.e., the latency between traffic requests' origin and target physical nodes); this increase was approximately 100% on all NSDP variants (see Fig. 5.6c).

Applying different variants has also an important impact on the physical network. Since NSDP-ISFS and NSDP-ISSC variants allow sharing data-plane NFSs with other slices rather than the ones from the same initial slice request (which gives more flexibility to NFS placement decisions), different objective functions had opposite impacts on the physical network. Indeed, the NSDP approach demanded up to 40% more (resp. 50% less) bandwidth compared to the other variants when the number

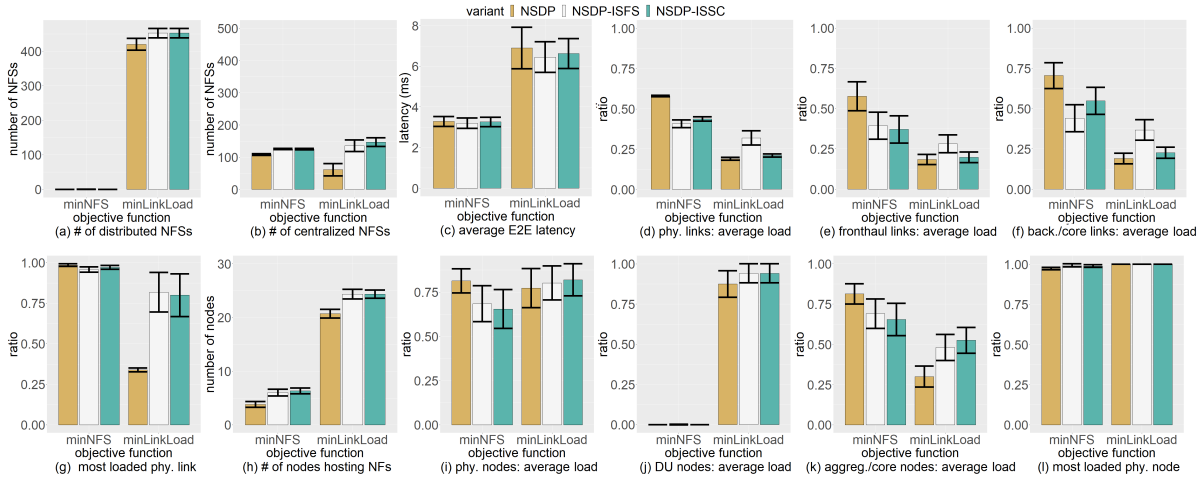


Figure 5.6 – Impact of different NSDP variants on the physical network.

of NFSs (resp. load on the physical links) is minimized (see Figures 5.6d, 5.6e, and 5.6f). Also, comparing the two objective functions, the load on physical links could be reduced by a factor of 3, and the most loaded physical link provided roughly 97% (resp. 34%) of its capacity when NSDP-ISSC (resp. NSDP) is applied along with minNFS (resp. minLinkLoad) objective function (see Fig. 5.6g). This behavior is explained by the concentration of NFSs on few physical nodes when the minNFS objective function is applied, hence stressing the related incoming links; applying the split setting 6 whenever is possible, this concentration is due to the greater number of centralized DP functions to be installed. Moreover, the NSDP-ISSC variant had a relevant impact on the physical nodes (see Fig. 5.6i); comparing this variant to NSDP and regarding the minNFS formulation, the load on physical nodes could be decreased from 85% to 65% on average.

NFSs' compression coefficients also play an important role in the final solution. Indeed, depending on the parameter to be optimized, different split settings are prioritized. For instance, when the load on the physical links is minimized, split setting 1 is always selected when it is admissible by DUs' capacity (see Fig. 5.6j); this functional split places all DP NFSs locally and therefore completely compresses the traffic demands' flow before sending it through fronthaul links (see Fig. 5.6e). It is worth mentioning that, regarding the three proposed physical topologies, we observed an important difference only on the end-to-end DP latency and on the fronthaul links' load. In our simulations, while the average end-to-end DP latency was 2.50 ms on the Tree structure, these values increased to 4.20 ms and 6.60 ms on Sun and Mandala topologies, respectively. Let us recall that, since there is only one possible elementary path to connect any pair of physical nodes on the Tree topology and slice

request 3 imposes a strict DP latency (see Tab. 5.2), the latency on the related physical links is lower than those found on Sun and Mandala structures; otherwise, some instances would be infeasible. For the same reason and in order to carry the flow from slice request 1, the links' capacity on Tree topology is greater than on the other two structures. Hence, when the number of NFS copies was minimized, the average load on fronthaul links on Tree, Sun, and Mandala was respectively 12.50%, 30%, and 40%. However, running the same instance on each of these topologies, we observed no difference in the selected split setting on the final solutions. However, this behavior might not be observed in real scenarios since physical networks are unlikely to have enough capacity to allow any split setting and hence be able to allocate resources to all slice requests. This, therefore, reinforces the importance of applying flexible functional splitting in future 5G systems and beyond.

5.7 Summary

In this chapter, we modeled the network slice provisioning as an optimization problem including novel mapping and provisioning requirements. In particular, we considered novel mapping dimensions appearing with 5G systems, modeling the relationship between flexible radio access functional splitting, control-plane and data-plane function separation, and sharing policies. Different variants of the problem were also proposed and the related models are compliant with running standards. We demonstrated by simulation the impact of taking into full and partial consideration of the peculiar constraints rising from the standards. For instance, we reported numerical results showing that flexible splitting appears as a key factor to deal with heterogeneous requirements to deploy distinct communication services, leading to considerable network slice cost decrease. In our simulations, the number of NFSs needed to deploy the virtual networks could be reduced by up to 56% depending on which of the six proposed sharing policies is applied to each network slice. We also observed that different variants related to the flexible splitting have an important impact on the physical network; depending on the selected approach, the average load on physical links could be reduced by a factor of 3.

Chapter 6

Exact approaches for the NSDP

Contenu

In this chapter, we propose several exact approaches based on the MILP formulation (5.1)-(5.19) for the problem, which includes novel *splitting*, *mapping* and *provisioning* constraints described in the published 5G standards documents [6, 7, 8]. We propose several classes of valid inequalities in order to strengthen the linear relaxation of the proposed MILP and integrate them in a Branch-and-Cut framework to solve the problem. We further present several strategies to reduce the symmetries and the size of the model[1]. All notations follow those presented in Table 5.1.

1. The content of this chapter was submitted to Discrete Applied Mathematics Journal as:
W. d. S. Coelho, A. Benhamiche, N. Perrot and S. Secci, "Exact Approaches for the Network Slice Design Problem", September 2021.

6.1 Symmetry-breaking constraints

This formulation could be strengthened by eliminating the inherent symmetries corresponding to the NF usage. The following inequalities are symmetry-breaking constraints for the NSDP.

$$x_{nu}^{sf} \leq \sum_{t \in S} \sum_{g \in F} \sum_{v \in V} x_{n-1v}^{tg} \qquad \forall s \in S, \forall f \in F, \forall u \in V, \forall n \in N \backslash \{n_1\} \tag{6.1}$$

Inequalities (6.1) allow assigning the NFs in an ordered way, by forbidding the use of an NF $n+1$ if NF n is available (i.e., hosting no NFS).

Proof. Given any NF $n \in N$, having the left-hand side of inequality (6.1) equals to zero (i.e., $x_{nu}^{sf} = 0$ for any $s \in S, f \in F, u \in V_p$) implies that n was not assigned to any $f \in F$ and hence it is available to host any NFS from any slice request. Since we check this feature in a paired, ordered way (i.e., $(n-1, n)$) and there is no utilization cost related to the uncapacitated functions from N, the right-hand side of the same inequality can hold any value equal or greater than zero; the related upper-bound will be imposed by constraints (5.4)-(5.13). Hence, (6.1) cut off part of the feasible region, while guaranteeing that at least one optimal solution from the original problem remains feasible. □

Example: Consider an instance with one slice request s_1 and three available network functions $\{n_1, n_2, n_3\}$ to host two different NFS types $\{f_1, f_2\}$. Given a physical node u_1 and regardless the related capacity constraints, all the following values for the x variables would imply the same cost in the complete solution for the problem:

$$x_{n_1u_1}^{s_1f_1} = x_{n_2u_1}^{s_1f_2} = 1 \qquad \text{and} \qquad x_{n_2u_1}^{s_1f_1} = x_{n_3u_1}^{s_1f_1} = x_{n_1u_1}^{s_1f_2} = x_{n_3u_1}^{s_1f_2} = 0 \tag{6.2}$$

$$x_{n_1u_1}^{s_1f_1} = x_{n_3u_1}^{s_1f_2} = 1 \qquad \text{and} \qquad x_{n_2u_1}^{s_1f_1} = x_{n_3u_1}^{s_1f_1} = x_{n_1u_1}^{s_1f_2} = x_{n_2u_1}^{s_1f_2} = 0 \tag{6.3}$$

$$x_{n_2u_1}^{s_1f_1} = x_{n_3u_1}^{s_1f_2} = 1 \qquad \text{and} \qquad x_{n_1u_1}^{s_1f_1} = x_{n_3u_1}^{s_1f_1} = x_{n_1u_1}^{s_1f_2} = x_{n_3u_1}^{s_1f_2} = 0 \tag{6.4}$$

However, applying Proposition 6.1, only (6.2) would be feasible.

6.2 Valid inequalities

We now present several families of valid inequalities used to strengthen the linear relaxation of formulation (5.1)-(5.19).

6.2.1 Lower-bound inequality

The first inequality expresses a lower bound on the number of NFSs needed to satisfy all the slice requests of S and the associated traffic demands.

The following inequality

$$\sum_{f \in F^c} \left\lceil \sum_{s \in S} \frac{n_s b_f}{cap(f)} \right\rceil + \sum_{f \in F^d} \left\lceil \sum_{k \in K(s): s \in S} \frac{\lambda_{f-1} b^k}{cap(f)} \right\rceil \leq \sum_{f \in F} \sum_{n \in N} \sum_{v \in V} y_{nu}^f \tag{6.5}$$

is valid for the NSDP.

Proof. The left-hand side of inequality (6.5) is the sum of constraints (5.2) when isolation (5.4)-(5.5) constraints and capacity inequalities (5.12)-(5.13) are relaxed. Hence, this is the minimum value for the right-hand side of inequality (6.5), which is the sum of constraints-(5.3) and the optimized parameter in the objective function (7.9) disregarding the required capacities and their related costs. □

6.2.2 Shortest path-based inequalities

The following trivial valid inequalities are based on the fact that the physical path assigned to carry the flow associated to a traffic demand k cannot be shorter than the shortest path between o_k and t_k that is capable of carry the flow of k completely compressed. For each traffic demand $k \in K(s) : s \in S$, let $sp(k)$ be the end-to-end latency on the shortest path between o_k and t_k with enough available capacity to carry the flow of k. Then, the following inequalities

$$\sum_{a \in A_p} d_a (\gamma^{ka}_{f_{|F^d|} f_0} + \sum_{f \in \{f_0\} \cup F^d \setminus \{f_{|F^d|}\}} \gamma^{ka}_{f f+1}) \geq sp(k) \qquad , \forall k \in K(s) : s \in S \tag{6.6}$$

are valid for the NSDP problem.

Proof. Given a traffic demand $k \in K$, the left-hand side of inequalities (6.6) contains all possible gamma variables that will be present in any feasible solution for an NSDP instance; due to the flow conservation constraints (5.9), the variables holding 1 represent the path connecting the origin and target nodes of k and transversing all installed DP NFSs copies related to the same traffic demand. Clearly, the sum of latency of all active arcs carrying the traffic sent by k cannot be smaller than the end-to-end latency on the shortest path with enough capacity to carry the expected volume between the related origin and target nodes. □

(a) Without SP valid inequality. (b) With SP valid inequality.

Figure 6.1 – An example of applying the shortest path-based (SP) valid inequality.

Example: Let $G = (V, A)$ be the graph representing the physical network, where $V = \{u1, u2\}$ is the set of nodes and $A = \{a1, a2\}$ is the set of arcs in G. Note that both arcs have their origin in node $u1$ and their destination in node $u2$; their available bandwidth and latency are showed in Fig. 6.1. Also, consider a scenario with only one NFS f_1 installed on the origin node of the traffic demand k_1, whose expected volume from its origin node o_k to its target node t_k is equals to 100Gbs after being compressed by NFS f_1; the dummy function f_0 is installed in the target node t_k as previously discussed. As depicted by Fig. 6.1b, $\gamma_{f_1 f_0}^{k_1 a1} = \gamma_{f_1 f_0}^{k_1 a2} = 0.50$ is a feasible solution since it respects all technical constraints for the linear relaxation of the formulation (5.1)-(5.19). Since only arc $a2$ is capable of carry the complete flow of k_1, $sp(k_1)$ is equal to 100ms. By Proposition 6.2.2, this fractional solution is no longer feasible since we add the following valid inequality:

$$10\gamma_{f_1 f_0}^{k_1 a2} + 100\gamma_{f_1 f_0}^{k_1 a2} \geq 100 \tag{6.7}$$

Hence, only setting $\gamma_{f_1 f_0}^{k_1 a1}$ to 0.00 and $\gamma_{f_1 f_0}^{k_1 a2}$ to 1.00 is a feasible solution as shown in Fig. 6.1b.

6.2.3 Minimum cut-based inequalities

The following valid inequalities are based on one min-cut max-flow theorem of Ford and Fulkerson [120], which states that for a single commodity, the maximum flow is equal to the minimum cut separating the related origin and target nodes. We, therefore, consider only the arcs with enough capacity to carry the expected flow individually. For each $k \in K(s) : s \in S$, let $\Delta(k)$ be the set of all minimum cuts separating o_k from t_k, such that for any $\delta \in \Delta(k) \subseteq A_p$, $\delta = \{a | b_a \geq \lambda_{f_{|F^d|}} b_k\}$. Then, the following inequalities

$$\sum_{a \in \delta} (\gamma_{f_{|F^d|} f_0}^{ka} + \sum_{f \in \{f_0\} \cup F^d \setminus \{f_{|F^d|}\}} \gamma_{f f+1}^{ka}) \geq 1 \qquad \forall k \in K(s) : s \in S, \forall \delta \in \Delta(k) \tag{6.8}$$

are valid for the NSDP problem.

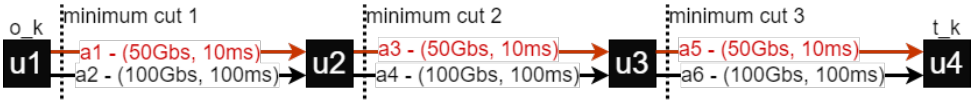


Figure 6.2 – An example of applying the minimum cut-based valid inequality.

Proof. For any $k \in K(s) : s \in S$, let O_k and T_k be two disjoint sets such that $O_k \cup T_k = V_p$, $O_k \cap T_k = \emptyset$, $o_k \in O_k$, and $t_k \in T_k$. Now, let δ_k be a minimum cut separating O_k from T_k such that $\delta_k \subseteq A_p$ and for any arc $(u, v) \in \delta_k$, $(b_a \geq \lambda_{f_{|F^d|}} b_k)$ holds and exactly one of its extremities is in O_k, implicating $(u \in T_k) \oplus (v \in T_k)$. Clearly, removing δ_k from A_p, there would no longer exist a path connecting o_k to t_k and capable of carry the expected volume from traffic demand k. Hence, a feasible solution for NSDP problem must use at least one arc from δ_k, whose capacity must be greater or equal to the compressed flow related to such traffic demand k. □

Example: Let $G = (V, A)$ be the graph representing the physical network, where $V = \{u1, .., a4\}$ is the set of nodes and $A = \{a1, .., a6\}$ is the set of arcs whose available bandwidth and latency are showed in Fig. 6.2. Also, consider a scenario with only one NFS f_1 installed on the origin node of the traffic demand k_1, whose expected volume from its origin node $o_k = u1$ to its target node $t_k = u4$ is equals to 100Gbs after being compressed by NFS f_1; the dummy function f_0 is installed in the target node t_k as previously discussed. By setting all γ variables to 0.50 is a feasible solution for the linear relaxation of the formulation (5.1)-(5.19) since it respects all technical constraints: the flow of k_1 is then split and each arc carries only 50Gbs. Applying Fulkerson [120] theorem, we have three different minimum cuts: $(a1, a2)$, $(a3, a4)$, and $(a5, a6)$. However, only arcs $a2$, $a4$, and $a6$ are capable of carry the complete flow of k_1 individually. Consequently, by Proposition 6.2.3, the initial fractional solution is no longer feasible since we add the following valid inequalities:

$$\gamma_{f_1 f_0}^{k_1 a2} \geq 1 \tag{6.9}$$

$$\gamma_{f_1 f_0}^{k_1 a4} \geq 1 \tag{6.10}$$

$$\gamma_{f_1 f_0}^{k_1 a6} \geq 1 \tag{6.11}$$

Hence, only setting γ_{f1f0}^{k1a2}, $\gamma_{f_1 f_0}^{k_1 a4}$, $\gamma_{f_1 f_0}^{k_1 a4}$ to 1.00 and all other variables to zero is a feasible solution.

6.2.4 Clique-based inequalities

In what follows, we introduce a set of inequalities based on the so-called *conflict graphs* related to the isolation constraints on one hand and the physical capacities on the other hand.

6.2.4.1 Virtual isolation

Let $\mathcal{H}_n = (V_n, E_n)$ be the associated graph obtained for each function $n \in N$ as follows. A node v in V_n is associated with every tuple $(s, f, u) : s \in S, f \in F, u \in V$ and there exists an edge in $(u, v) \in E_n$ between any two nodes $u = (s, f, u)$ and $v = (t, g, u)$ from V_n if $q^{st}_{fg} * q^{ts}_{gf} = 0$ holds. Hence, an edge in $\mathcal{H}_n$ exists if two NFSs are packed together in the same function n while violating the isolation constraints imposed by the NSs.

Let $\mathcal{C}(\mathcal{H}_n)$ be the set of all cliques in $\mathcal{H}_n$. Then, the following inequalities

$$\sum_{(s,f,u)\in\mathcal{C}} x^{sf}_{nu} \leqslant 1 \qquad \forall n \in N, \forall \mathcal{C} \in \mathcal{C}(\mathcal{H}_n) \tag{6.12}$$

are valid for the NSDP.

Proof. Let $\mathcal{C}$ be a clique in $\mathcal{C}(\mathcal{H}_n)$. If two tuples (s, f, u) and (t, g, v) from $\mathcal{C}$ are mapped to a same network function $n \in N$, the related isolation constraints (5.4) will be violated. I other words, each pair of tuples from clique $\mathcal{C}$ represents an infeasible packing of NFSs from different slices requests in an NF $n \in N$. Consequently, a clique in the graph $\mathcal{H}_n$ corresponds to a set of NFS that cannot be simultaneously packed in any network function $n \in N$. □

Example: Consider a small instance with three slice requests $\{s_1, s_2, s_3\}$, one CP NFS type f_c, three NF $\{n_1, n_2, n_3\}$ and one non-access node u_1 in the physical network. For this instance, we also have $q^{st}_{f_c f_c} * q^{ts}_{f_c f_c} = 0$ for any pair (s,t) of distinct slice requests from S. From (5.4), we have:

$$x^{s_1 f_c}_{n_1 u_1} + x^{s_2 f_c}_{n_1 u_1} + x^{s_3 f_c}_{n_1 u_1} + x^{s_1 f_c}_{n_2 u_1} + x^{s_2 f_c}_{n_2 u_1} + x^{s_3 f_c}_{n_2 u_1} + x^{s_1 f_c}_{n_3 u_1} + x^{s_2 f_c}_{n_3 u_1} + x^{s_3 f_c}_{n_3 u_1} \leq 4.50 \tag{6.13}$$

Now, consider the linear relaxation of the problem in which we no longer have the integrity constraints on the variables. Setting 0.5 all x variables gives a feasible fractional solution for (6.13) and for the whole relaxed model. By Proposition 6.2.4.1, this fractional solution is no longer feasible since

we add the following aggregated cut:

$$x_{n_1u_1}^{s_1f_c} + x_{n_1u_1}^{s_2f_c} + x_{n_1u_1}^{s_3f_c} + x_{n_2u_1}^{s_1f_c} + x_{n_2u_1}^{s_2f_c} + x_{n_2u_1}^{s_3f_c} + x_{n_3u_1}^{s_1f_c} + x_{n_3u_1}^{s_2f_c} + x_{n_3u_1}^{s_3f_c} \leq 3 \tag{6.14}$$

6.2.4.2 Physical isolation

Let $\mathcal{H}_u = (V_u,\ E_u)$ be the associated undirected graph obtained from an instance of NSDP as follows. A node v in V_u is associated with every tuple $(s, f, n) : s \in S, f \in F, n \in N$ that $\bar{x}_{nu}^{sf} > 0$ holds in the current solution. In addition, there exists an edge in $(u, v) \in E$ between any two nodes $u = (s, f, n)$ and $v = (t, g, n)$ from V_u if $(q^{st} * q^{ts} = 0)$ holds. In other words, an edge in $\mathcal{H}_u$ exists if two functions are packed together in any node $u \in V_p$ while violating the isolation constraints imposed by the related slices. Let $\mathcal{C}(\mathcal{H}_u)$ be the set of all cliques on $\mathcal{H}_u$. Then, the following inequalities

$$\sum_{(s,f,n)\in\mathcal{C}} x_{nu}^{sf} \leqslant 1 \qquad\qquad \forall u \in V_p^{na}, \forall \mathcal{C} \in \mathcal{C}(\mathcal{H}_u) \tag{6.15}$$

are valid for the NSDP.

Proof. Let $\mathcal{C}$ be a clique in $\mathcal{C}(\mathcal{H}_u)$. If two tuples (s, f, n) and (t, g, n) from $\mathcal{C}$ are assigned to a same physical node $u \in V_p$, the related isolation constraints (5.5) will be violated. I other words, each pair of tuples from clique $\mathcal{C}$ represents an infeasible packing of NFSs from different slices requests in a node $u \in V_p$. Consequently, a clique in the graph $\mathcal{H}_u$ corresponds to a set of NFS that cannot be simultaneously installed in any physical node $u \in V_p$. □

Example: Consider a small instance with three slice requests $\{s_1, s_2, s_3\}$, one CP NFS type f_c, three NF $\{n_1, n_2, n_3\}$ and one non-access node u_1 in the physical network. For this instance, we also have $q^{st} * q^{ts} = 0$ for any pair (s,t) of distinct slice requests from S. From (5.5), we have:

$$x_{n_1u_1}^{s_1f_c} + x_{n_1u_1}^{s_2f_c} + x_{n_1u_1}^{s_3f_c} + x_{n_2u_1}^{s_1f_c} + x_{n_2u_1}^{s_2f_c} + x_{n_2u_1}^{s_3f_c} + x_{n_3u_1}^{s_1f_c} + x_{n_3u_1}^{s_2f_c} + x_{n_3u_1}^{s_3f_c} \leq 4.50 \tag{6.16}$$

$$x_{n_1u_2}^{s_1f_c} + x_{n_1u_2}^{s_2f_c} + x_{n_1u_2}^{s_3f_c} + x_{n_2u_2}^{s_1f_c} + x_{n_2u_2}^{s_2f_c} + x_{n_2u_2}^{s_3f_c} + x_{n_3u_2}^{s_1f_c} + x_{n_3u_2}^{s_2f_c} + x_{n_3u_2}^{s_3f_c} \leq 4.50 \tag{6.17}$$

$$x_{n_1u_3}^{s_1f_c} + x_{n_1u_3}^{s_2f_c} + x_{n_1u_3}^{s_3f_c} + x_{n_2u_3}^{s_1f_c} + x_{n_2u_3}^{s_2f_c} + x_{n_2u_3}^{s_3f_c} + x_{n_3u_3}^{s_1f_c} + x_{n_3u_3}^{s_2f_c} + x_{n_3u_3}^{s_3f_c} \leq 4.50 \tag{6.18}$$

Now, consider the linear relaxation of the problem in which we no longer have the integrity constraints on the variables. Setting 0.5 all x variables gives a feasible fractional solution for (6.16)-(6.18)

and for the whole relaxed model. However, by applying Proposition 6.2.4.2, this fractional solution is no longer feasible since we add the following aggregated cuts:

$$x^{s_1f_c}_{n_1u_1} + x^{s_2f_c}_{n_1u_1} + x^{s_3f_c}_{n_1u_1} + x^{s_1f_c}_{n_2u_1} + x^{s_2f_c}_{n_2u_1} + x^{s_3f_c}_{n_2u_1} + x^{s_1f_c}_{n_3u_1} + x^{s_2f_c}_{n_3u_1} + x^{s_3f_c}_{n_3u_1} \leq 3 \tag{6.19}$$

$$x^{s_1f_c}_{n_1u_2} + x^{s_2f_c}_{n_1u_2} + x^{s_3f_c}_{n_1u_2} + x^{s_1f_c}_{n_2u_2} + x^{s_2f_c}_{n_2u_2} + x^{s_3f_c}_{n_2u_2} + x^{s_1f_c}_{n_3u_2} + x^{s_2f_c}_{n_3u_2} + x^{s_3f_c}_{n_3u_2} \leq 3 \tag{6.20}$$

$$x^{s_1f_c}_{n_1u_3} + x^{s_2f_c}_{n_1u_3} + x^{s_3f_c}_{n_1u_3} + x^{s_1f_c}_{n_2u_3} + x^{s_2f_c}_{n_2u_3} + x^{s_3f_c}_{n_2u_3} + x^{s_1f_c}_{n_3u_3} + x^{s_2f_c}_{n_3u_3} + x^{s_3f_c}_{n_3u_3} \leq 3 \tag{6.21}$$

6.2.4.3 Link capacity

Let $\mathcal{H}_a = (V_a,\ E_a)$ be the associated graph obtained for each physical link $a \in A$ as follows. A node v in V_a is associated with every tuple $(k, f, g) : s \in S, k \in K(s), f, g \in F$. Let $b(k, f, g)$ be the total data flow between NFSs f and g from commodity k. There exists an edge in $(u, v) \in E_a$ between any two nodes $u = (k, f, g)$ and $v = (k^{'}, f^{'}, g^{'})$ from V_a if $b(k, f, g) + b(k^{'}, f^{'}, g^{'}) > b_a$ holds. Let $\mathcal{C}(\mathcal{H}_a)$ be the set of all cliques on $\mathcal{H}_a$. Then, the following inequalities

$$\sum_{(k,f,g)\in\mathcal{C}} \gamma^{ka}_{fg} \leqslant 1 \qquad\qquad \forall a \in A, \forall \mathcal{C} \in \mathcal{C}(\mathcal{H}_a) \tag{6.22}$$

are valid for the NSDP.

Proof. Let $\mathcal{C}$ be a clique in $\mathcal{C}(\mathcal{H}_a)$. If the flow of two pairs (k, f, g) and $(k^{'}, f^{'}, g^{'})$ from $\mathcal{C}$ are carried through a same physical link $a \in A_p$, the related capacity constraint (5.12) will be violated. I other words, each pair of nodes from clique $\mathcal{C}$ represents an infeasible routing from different traffic demands in a physical link $a \in A_p$. Consequently, a clique in the graph $\mathcal{H}_a$ corresponds to a set of traffic demands that cannot be carried simultaneously through physical link a. □

Example: Consider a small instance with one slice requests s_1 with two traffic demands $\{k_1, k_2\}$, one DP NFS type f_1, f_2, and several physical arcs $\{a_1, .., a_{|A_p|}\}$ in the physical network. For this instance, we also have $b_a = 15$ for any $a \in A_p$ and $b_{k_1} = b_{k_2} = 10$. From (5.12), we have:

$$10\gamma^{k_1a}_{f_0f_1} + 10\gamma^{k_2a}_{f_0f_1} \leq 15 \qquad\qquad ,\forall a \in A_p \tag{6.23}$$

This constraint imposes that the volume sent by each traffic demand to the DP NFS f_1 and carried by the arc a cannot be greater than its capacity. Now, consider the linear relaxation of the problem

in which we no longer have the integrity constraints on the variables. Setting 0.75 all γ variables gives a feasible fractional solution for (6.23). However, by applying Proposition 6.2.4.3, this fractional solution is no longer feasible since we add the following cuts:

$$\gamma_{f_0 f_1}^{k_1 a} + \gamma_{f_0 f_1}^{k_2 a} \leq 1 \qquad ,\forall a \in A_p \tag{6.24}$$

6.2.4.4 Node capacity

This class of valid inequalities depends directly on the values of x and w variables and therefore can be generated only regarding a solution for the presented formulation (5.1)-(7.9). Let $\bar{w}_{nu}^{sf}$ and $\bar{x}_{nu}^{sf}$ be the values of the related variables in the current solution. For each $u \in V_p$, let $\mathcal{H}_u = (V_u, E_u)$ be the associated undirected graph obtained from an instance of NSDP as follows. A node v in V_u is associated with every tuple $(s, f, n) : s \in S, f \in F, n \in N$ that $\bar{x}_{nu}^{sf} > 0$ holds in the current solution. There exists an edge in $(u, v) \in E_u$ between any two nodes $u = (s, f, n)$ and $v = (t, g, m)$ from V_u if $\frac{\bar{w}_{nu}^{sf}}{\bar{x}_{nu}^{sf}} c_f^c + \frac{\bar{w}_{mu}^{tg}}{\bar{x}_{mu}^{tg}} c_g^c > c_u^c$ holds for any physical resource $c \in C$. Let $\mathcal{C}(\mathcal{H}_u)$ be the set of all cliques on $\mathcal{H}_u$. Then, the following inequalities

$$\sum_{(s,f,n)\in\mathcal{C}} x_{nu}^{sf} \leqslant 1 \qquad ,\forall u \in V_p, \forall \mathcal{C} \in \mathcal{C}(\mathcal{H}_u) \tag{6.25}$$

are valid for the NSDP.

Proof. Let $\mathcal{C}$ be a clique in $\mathcal{C}(\mathcal{H}_u)$. If the flow of two pairs (s, f, n) and (t, g, m) from $\mathcal{C}$ are carried through a same physical node $u \in V_p$, the related capacity constraint (5.13) will be violated. I other words, each pair of nodes from clique $\mathcal{C}$ represents an infeasible packing from different NFSs in a physical node $u \in V_p$. Consequently, a clique in the graph $\mathcal{H}_u$ corresponds to a set of functions that cannot be packed simultaneously in the physical node u. □

Example: Consider a small instance with two slice requests $\{s_1, s_2\}$, one DP NFS type f_1, two available NFs $\{n_1, n_2\}$, and two aggregation/core nodes $\{u_1, u_2\}$ with only one resource type c_1 in the physical network. For this instance, we also have $c_u^1 = 15$ for any $u \in V_p^{ac}$, $\frac{n_1 b_{f_1}}{cap(f_1)} = \frac{n_2 b_{f_1}}{cap(f_1)} = 1$, and $c_{f_1}^1 = 10$. From (5.2),(5.3), and (5.13), we have:

$$w_{nu}^{sf_1} = x_{nu}^{sf_1} \qquad ,\forall s \in S, \forall n \in N, \forall u \in V_p^{ac} \tag{6.26}$$

$$w_{nu}^{s_1 f_1} + w_{nu}^{s_2 f_1} \leq y_{nu}^{f_1} \leq 1.5 \qquad ,\forall n \in N, \forall u \in V_p^{ac} \tag{6.27}$$

Now, consider the linear relaxation of the problem in which we no longer have the integrity constraints on the variables. Setting $x_{n_1u_1}^{s_1f_1}$ and $x_{n_2u_1}^{s_2f_1}$ to 0.75 gives a feasible fractional solution for (6.26) and (6.27). However, by applying Proposition 6.2.4.4, this fractional solution is no longer feasible since the following cut are added:

$$x_{n_1u_1}^{s_1f_1} + x_{n_2u_1}^{s_2f_1} \leq 1 \qquad ,\forall n \in N, \forall u \in V_p^{ac} \tag{6.28}$$

6.3 A branch-and-cut algorithm for the NSDP

In this work, we first propose a parallelized Branch-and-Cut algorithm to solve the problem efficiently. In this algorithm, referred to as *Clique-based Branch-and-Cut* (CBC), we build the conflict graphs in each node of the branch-and-bound tree as follows.

Physical node isolation: A node in the related conflict graph $\mathcal{H}_u = (V_u, E_u)$ is associated with every tuple $(s, f, n) : s \in S, f \in F, n \in N$ that $\bar{x}_{nu}^{sf} > 0$ holds in the current solution. In addition, there exists an edge in E_u between any two nodes (s, f, n) and (t, g, n) from V_u if $(q^{st} * q^{ts} = 0) \wedge (\bar{x}_{nu}^{sf} + \bar{x}_{nu}^{tf} > 1)$ holds[2]. In other words, given a fractional solution, an edge in $\mathcal{H}_u$ exists if two functions are packed together in any node $u \in V_p$ while violating the isolation constraints imposed by the related slices.

Network function isolation: For each $n \in N$, a node in the related conflict graph $\mathcal{H}_n = (V_n, E_n)$ is associated with every tuple $(s, f, u) : s \in S, f \in F, u \in V_p$ that $\bar{x}_{nu}^{sf} > 0$ holds in the current solution. Also, there exists an edge in E_n between any two nodes (s, f, u) and (t, g, u) from V_n if $(q_{fg}^{st} * q_{gf}^{ts} = 0) \wedge (\bar{x}_{nu}^{sf} + \bar{x}_{nu}^{tf} > 1)$ holds.

Physical link capacity: A node in the related conflict graph is associated with every tuple $(k, f, g) : s \in S, k \in K(s), f, g \in F$ that $\bar{\gamma}_{fg}^{ka} > 0$ holds in the current solution. Considering different traffic generation dependencies[3] and the compression coefficients of each NFS type, let $b(k, f, g)$ be the total data flow between NFSs f and g from commodity k. There exists an edge in E_a between any two nodes $u = (k, f, g)$ and $v = (k', f', g')$ from V_a if $(b(k, f, g) + b(k', f', g') > b_a) \wedge (\bar{\gamma}_{f,g}^{k,a} + \bar{\gamma}_{f',g'}^{k',a} > 1)$ holds.

2. It is worth mentioning that, if $s = t$, $q^{st} + q^{ts} = 2$ always holds.
3. As previously discussed, data-plane traffic depends on the service is provided by the slice while control-plane traffic relies on the expected number of UE and their behaviors.

Physical node capacity: Let $\bar{w}^{sf}_{nu}$ and $\bar{x}^{sf}_{nu}$ be the values of the related variables in the current solution. For each physical node $u \in V$, a node in the related conflict graph $\mathcal{H}_u = (V_u, E_u)$ is associated with every tuple $(s, f, n) : s \in S, f \in F, n \in N$ that $\bar{x}^{sf}_{nu} > 0$ holds in the current solution. There exists an edge in E_u between any two nodes $u = (s, f, n)$ and $v = (t, g, m)$ from V_u if $\frac{\bar{w}^{sf}_{nu}}{\bar{x}^{sf}_{nu}} c^c_f + \frac{\bar{w}^{tg}_{mu}}{\bar{x}^{tg}_{mu}} c^c_g > c^c_u$ holds for any physical resource $c \in C$.

Regarding the solution of the current LP, we run several separation routines in parallel as follow: each available thread is responsible for creating a conflict graph from the capacity and isolation constraints, and identifying violated clique inequalities (6.12)-(6.25), as previously discussed. All cliques provided by the separation routines are found by using Bron-Kerbosch algorithm [121] and added as *user cuts* through the solver's callback procedure [122].

6.4 A row-generation framework for the NSDP

We also propose the *Reduced MILP-based Row-Generation* (RMRG) framework, which is based on a relaxed model: the original formulation (5.1)-(5.19) is reduced to a new one without the isolation, capacity, and latency constraints. Hence this new formulation has only the constraints related to the main decisions on the NSDP: the split selection inequalities (5.1), the dimensioning equations (5.2), the packing inequalities (5.3), the placement constraints (5.7) and (8.27), and the routing constraints (5.9); integrality constraints remain in the model. This reduced model can thereby be considerably smaller, having up to 98% fewer constraints compared to the original formulation; for the instances with only one element in each input set, the reduced model has 50% fewer constraints. We then denote by L the set of relaxed constraints, namely the isolation constraints (5.4)-(5.6), the capacity inequalities (5.13) and (5.19), and the latency constraints (5.10) and (5.11). During the branch-and-bound process, the *Lazy Constraints* routine (available in the MILP solver's callback procedure [122]) is called each time the solver finds a new and better integer solution: if the current solution violates any constraint from L, it is added as *cut* to the reduced model. We developed a parallelized framework in which each parallel process is responsible for searching and adding the violated constraints related to a given physical node or link.

Table 6.1 – Instance Sizes

Instance size	$\|V\|$	Graph density	$\|S\|$	Demands per slice	$\|F^d\|$	$\|F^c\|$
Tiny (T)	10	0.15	2	1	2	2
Small (S)	15	0.10	2	2	4	2
Medium (M)	20	variable	4	3	4	3

Table 6.2 – Instance Classes

Latency	Description
Low (L)	The maximum latency d_{fg} between two NFSs and the end-to-end latency d_s imposed by each slice request $s \in S$ is set respectively to between 50% and 150% and to between 250% and 500% of the average latency on the physical links.
High (H)	The maximum latency d_{fg} between two NFSs and the end-to-end latency d_s of each slice $s \in S$ is set respectively to between 200% and 400% and to between 300% and 1000% of the average latency on the physical links.
Capacity	**Description**
Tight (T)	The available bandwidth b_a on the physical links have between 50% and 100% of the average volume (without compression) generated by the slices. In addition, each physical node $u \in V \backslash V^{ap}$ has enough capacity to host between 1 and 3 copies of each NFS type; application nodes has no available capacity.
Moderate (M)	The available bandwidth b_a on the physical links have between 200% and 300% of the average volume (without compression) generated by the slices. In addition, each physical node $u \in V \backslash V^{ap}$ has enough capacity to host between 5 and 8 copies of each NFS type; application nodes has no available capacity.
Isolation	**Description**
Weak (W)	10% of isolation parameters q^{st} and q^{st}_{fg} are set to 0; they are randomly chosen.
Strong (S)	75% of isolation parameters q^{st} and q^{st}_{fg} are set to 0; they are randomly chosen.

6.5 Numerical experiments

We generated different instance sizes (see Table 6.1), in which the processing capacity $cap(f)$ of each NFS in F^d was set to a value between 50% and 100% of the average volume generated by the traffic demands. For NFSs of F^c, this value was set to between 50% and 100% of the volume related to the total number n_s of expected UEs connected to the slice. Also, the total amount b_{fg} of traffic between two functions from $F(s) \cup G(s)$ was set to 1 Kbps per UE. As shown in Table 6.2, different instance classes are also proposed, which are related to the ratio between the resources required by the slices and those available on the physical network, and also between the latency on the physical links and the threshold imposed by slices and pairs of NFSs. The complete reference of each generated instance is given by joining the acronyms of each size and class name from Table 6.1 and Table 6.2. For example, $< S, L, M, S >$ refers to a *small* instance with *low* latency threshold, *moderate* capacity requirements, and *strong* isolation constraints. The data-set and the source code are available on [123].

We implemented our model in a Julia-JuMP environment using ILO CPLEX 12.10 as the linear solver. Our tests were run on a Linux server with an Intel Xeon E5-2650 CPU. Finally, all tests were made by replacing the objective function (7.9) by the following one:

$$min \sum_{f \in F} \sum_{n \in N} \sum_{u \in V} \sum_{c \in C} c_f^c \mu_u^c y_{nu}^f \tag{6.29}$$

6.5.1 Model strengthening

We now analyze the impact of Symmetry-Breaking (SB) constraints (6.1), the Lower-Bound (LB) inequality (6.5), Shortest-Path (SP) constraints (6.6), and Min-Cut (MC) inequalities (6.8) on the original model (5.1)-(5.19). While each shortest path $sp(k)$ on (6.6) is calculated using Dijkstra's algorithm [19], we provide different minimum cuts to the set $\Delta(k)$ on (6.8) by using Edmonds-Karp algorithm [124], Dinic's algorithm [125], and Boykov-Kolmogorov algorithm [126], which are efficient algorithms to solve the related dual Max-Flow problem. We then eliminate all arcs that do not have enough capacity to individually carry the flow of the related traffic demand. Note that, by using these three different algorithms, we can provide at most three different minimum cuts. Table 6.3 and Table 6.4 show the impact of these different models with and without the proposed inequalities. The first and second columns refer to the instance and the additional inequalities (if any), respectively. While the third column shows the gap between the linear relaxation and the best integer solution found in one-hour runtime, the fourth column depicts the number of nodes on the branch-and-bound tree. Finally, the two last columns refer to the final gap and the total runtime. In all columns, we show the average and standard deviation of the related parameter from 10 different instances of the same class.

We observe first that LB inequality had an important impact on all instance classes, especially on scenarios with weak isolation. For instance, the average gap from the linear relaxation and final gap were respectively equal to 3% and 0% (resp. 2% and 1%) on $< T, T, H, W >$ (resp. $< S, T, H, W >$) instances. As seen in Table 6.3, SB inequalities better performed on instances with strict capacity, latency, and isolation constraints: the average reduction on the tree size was equal to 57% and 83% on $< T, T, L, S >$ and $< S, T, L, S >$ instances, respectively.

Interestingly, SP and MC inequalities worked better on instances with strong isolation. For in-

Table 6.3 – Impact of different valid inequalities: tiny instances.

Instance	Valid Inequalities	Lin. Relax. Gap	Tree Size (x10^6)	Final Gap	Runtime (s)
<T,T,L,S>	None	0.37±0.01	3.0±0.7	0.12±0.02	3600
	Lower-Bound	**0.16±0.05**	**2.0±0.3**	**0.0**	1528±1087
	Min-Cut	0.37±0.01	**1.6±0.2**	**0.0**	**1268±1165**
	Shortest-Path	0.37±0.01	**1.6±0.3**	**0.0**	**1285±1157**
	Symmetry-Breaking	0.37±0.01	**1.3±0.1**	**0.0**	**1243±1178**
	All	**0.16±0.05**	**1.4±1.0**	**0.0**	1398±1087
<T,T,L,W>	None	0.33±0.04	4.5+1.5	0.22±0.07	3600
	Lower-Bound	**0.1±0.04**	**3.0±1.5**	**0.08±0.04**	3600
	Min-Cut	0.33±0.04	3.2±1.5	**0.12±0.6**	3600
	Shortest-Path	0.33±0.04	3.4±1.4	**0.17±0.8**	3600
	Symmetry-Breaking	0.33±0.04	3.3±1.4	**0.17±0.8**	3600
	All	**0.1±0.04**	**2.7±1.4**	**0.09±0.04**	3600
<T,T,H,W>	None	0.22±0.04	1.6±1.4	0.0	1465±1117
	Lower-Bound	**0.03±0.01**	**0.5±0.4**	0.0	**460±298**
	Min-Cut	0.22±0.04	**1.5±0.7**	0.0	**1250±685**
	Shortest-Path	0.22±0.04	**1.5±0.6**	0.0	**1282±876**
	Symmetry-Breaking	0.22±0.04	1.7±0.6	0.0	1426±354
	All	**0.03±0.01**	**1.1±0.9**	0.0	**1276±378**
<T,T,H,S>	None	0.33±0.03	3.0±1.1	0.12±0.09	3600
	Lower-Bound	**0.18±0.4**	4.8±0.6	**0.08±0.05**	3600
	Min-Cut	0.33±0.03	4.5±0.2	**0.11±0.07**	3600
	Shortest-Path	0.33±0.03	5.0+0.2	**0.09±0.05**	3600
	Symmetry-Breaking	0.33±0.03	3.1±0.9	**0.10±0.04**	3600
	All	**0.18±0.4**	3.0±0.6	**0.07±0.02**	3600
<T,M,L,S>	None	0.26±0.2	8.2±1.0	0.13±0.03	3600
	Lower-Bound	**0.11±0.5**	**4.7±0.3**	**0.04±0.02**	3600
	Min-Cut	0.26±0.2	**4.0±1.1**	**0.06±0.03**	3600
	Shortest-Path	0.26±0.2	**4.0±0.9**	**0.07±0.04**	3600
	Symmetry-Breaking	0.26±0.2	**3.3±1.2**	**0.08±0.04**	3600
	All	**0.11±0.5**	**4.1±0.1**	**0.08±0.04**	3600
<T,M,L,W>	None	0.26±0.03	4.3±1.6	0.05±0.03	3600
	Lower-Bound	**0.13±0.02**	4.0±0.9	0.07±0.05	3600
	Min-Cut	0.26±0.03	3.9±1.5	0.06±0.03	3600
	Shortest-Path	0.26±0.03	4.1±1.3	0.06+0.03	3600
	Symmetry-Breaking	0.26±0.03	**2.6±1.2**	**0.0**	**1964±978**
	All	**0.13±0.02**	**3.0±0.8**	**0.04±0.02**	3600
<T,M,H,W>	None	0.23±0.03	6.0±1.4	0	2896±526
	Lower-Bound	**0.08±0.05**	**2.5±0.8**	0.0	**2279±668**
	Min-Cut	0.23±0.03	**5.0±0.3**	0.09±0.03	3600
	Shortest-Path	0.23±0.03	**4.6±0.3**	0.08±0.04	3600
	Symmetry-Breaking	0.23±0.03	**4.5+0.1**	0.1±0.04	3600
	All	**0.08±0.05**	**3.3±0.3**	0.04±0.03	3600
<T,M,H,S>	None	0.28±0.02	2.8±1.2	0.07±0.02	3600
	Lower-Bound	**0.18±0.04**	4.5±1.0	**0.01±0.01**	3600
	Min-Cut	0.28±0.02	5.8±1.5	**0.02±0.01**	3600
	Shortest-Path	0.28±0.02	4.5±0.9	**0.02±0.01**	3600
	Symmetry-Breaking	0.28±0.02	4.6±1.2	0.05±0.05	3600
	All	**0.18±0.04**	3.3±0.7	**0.02±0.01**	3600

Table 6.4 – Impact of different valid inequalities: small instances.

Instance	Valid Inequalities	Lin. Relax. Gap	Tree Size (x10^6)	Final Gap	Runtime (s)
<S,T,L,S>	None	0.26±0.03	4.7±1.1	0.21±0.3	3600
	Lower-Bound	**0.17±0.02**	**1.7±0.9**	0.01±0.01	3600
	Min-Cut	0.26±0.03	**1.4±0.7**	**0.0**	**2839±761**
	Shortest-Path	0.26±0.03	**1.5±0.2**	**0.0**	3209±315
	Symmetry-Breaking	0.26±0.03	**0.8±0.3**	**0.0**	**2562±1037**
	All	**0.17±0.02**	**1.0±0.5**	**0.0**	**2669±930**
<S,T,L,W>	None	0.21±0.02	4.2±0.3	0.19±0.02	3600
	Lower-Bound	**0.07±0.03**	**1.7±0.6**	**0.05±0.02**	3600
	Min-Cut	0.21±0.02	1.9±1.0	0.12±0.03	3600
	Shortest-Path	0.21±0.02	1.8±1.3	0.12±0.03	3600
	Symmetry-Breaking	0.21±0.02	**1.4±1.3**	0.14±0.02	3600
	All	**0.07±0.03**	**1.4±1.0**	**0.06±0.03**	3600
<S,T,H,W>	None	0.35±0.12	1.6±0.4	0.3±0.1	3600
	Lower-Bound	**0.02±0.01**	1.5±0.9	**0.01±0.01**	3600
	Min-Cut	0.35±0.12	1.7±0.6	0.11±0.01	3600
	Shortest-Path	0.35±0.12	1.6±0.9	0.12±0.02	3600
	Symmetry-Breaking	0.35±0.12	**1.3±0.3**	0.13±0.02	3600
	All	**0.02±0.01**	**1.3±0.5**	**0.02±0.01**	3600
<S,T,H,S>	None	0.23±0.04	2.6±0.2	0.13±0.03	3600
	Lower-Bound	**0.11±0.04**	**1.7±0.2**	**0.08±0.03**	3600
	Min-Cut	0.23±0.04	1.8±0.1	0.14±0.04	3600
	Shortest-Path	0.23±0.04	1.8±0.2	0.15±0.03	3600
	Symmetry-Breaking	0.23±0.04	**1.3±1.2**	**0.12±0.04**	3600
	All	**0.11±0.04**	**1.0±0.3**	**0.0**	**2689±910**
<S,M,L,S>	None	0.16±0.02	1.9±0.8	0.14±0.02	3600
	Lower-Bound	**0.07±0.01**	1.8±1.1	**0.05±0.01**	3600
	Min-Cut	0.16±0.02	1.8±0.4	**0.06±0.02**	3600
	Shortest-Path	0.16±0.02	1.9±0.7	**0.06±0.02**	3600
	Symmetry-Breaking	0.16±0.02	1.4±0.7	**0.07±0.01**	3600
	All	**0.07±0.01**	**1.2±1.1**	**0.06±0.01**	3600
<S,M,L,W>	None	0.16±0.02	3.6±0.6	0.15±0.04	3600
	Lower-Bound	**0.07±0.01**	**1.7±0.8**	**0.06±0.01**	3600
	Min-Cut	0.16±0.02	**1.8±0.7**	**0.10±0.03**	3600
	Shortest-Path	0.16±0.02	**1.8±0.9**	0.12±0.03	3600
	Symmetry-Breaking	0.16±0.02	**1.3±0.7**	0.12±0.03	3600
	All	**0.07±0.01**	**1.2±0.4**	**0.06±0.01**	3600
<S,M,H,W>	None	0.33±0.04	3.2±0.7	0.12±0.04	3600
	Lower-Bound	**0.22±0.05**	**1.7±1.1**	0.21±0.05	3600
	Min-Cut	0.33±0.04	**1.7±0.7**	0.27±0.04	3600
	Shortest-Path	0.33±0.04	1.8±0.7	0.25±0.04	3600
	Symmetry-Breaking	0.33±0.04	**1.4±1.2**	0.27±0.04	3600
	All	**0.22±0.05**	**1.4±0.8**	0.21±0.05	3600
<S,M,H,S>	None	0.21±0.02	2.5±0.2	0.15±0.01	3600
	Lower-Bound	**0.09±0.01**	**1.6±0.3**	**0.06±0.01**	3600
	Min-Cut	0.21±0.02	1.7±0.2	0.14±0.01	3600
	Shortest-Path	0.21±0.02	1.7±0.2	0.14±0.01	3600
	Symmetry-Breaking	0.21±0.02	**1.1±0.1**	0.015±0.01	3600
	All	**0.09±0.01**	**1.2±0.2**	**0.07±0.01**	3600

stance, the average final gap on scenarios with weak and strong isolation constraints was respectively 15% (resp. 8%) and 9% (resp. 5%) on small (resp. tiny) instances. Since the possibilities of embedding NFs into physical nodes are decreased when strong isolation constraints are applied, these valid inequalities led to fewer fractional routing-related variables γ in the linear relaxation: we observed up to 30% fewer variables holding values in $]0, 1[$. This behavior might therefore have an important impact in selecting a branching strategy by the solver's black box in order to find integer solutions.

The proposed SB, LB, SP, and MC inequalities (6.1)-(6.8) outperformed the original model with no valid inequalities in almost all tests, specially on instances without strict latency constraints (e.g., $< S, T, H, S >$). Together, the final gap could be reduced from 30% to 2% on $< S, T, H, W >$ instances. It is worthwhile mentioning that the original model had its performance decreased when applied on instances with strict capacity instances: the average final gap was respectively equal to 22% and 30% on $< T, T, L, W >$ and $< S, T, H, W >$ instances.

6.5.2 Branch-and-cut algorithm

We now present the results from the numerical experiments applying the proposed CBC algorithm along with the isolation-based valid inequalities (6.12) and (6.15) (hereafter referred to as *Isolation Cuts*) and the capacity-based valid inequalities (6.22) and (6.25) (hereafter referred to as *Capacity Cuts*) to the original model (5.1)-(5.19). The separation routines were made as previously discussed and using 8 threads. Finally, SB, LB, SP, and MC inequalities (6.1)-(6.8) were not applied in these tests.

Fig. 6.3 depicts the results applying the proposed branch-and-cut frameworks. We ran 10 instances of each class and, for sake of clarity, only the average of each parameter is presented. First, both capacity and isolation cuts had smaller gaps in almost all tests, especially on instances with strict latency and capacity constraints. For instance, compared to the branch-and-bound framework without any cut, the average gap decreased from 22% to 13% (resp. from 21% to 14%) on $< T, T, L, W >$ (resp. $< S, T, L, S >$) when capacity (resp. isolation) cuts were applied (see Fig. 6.3a; resp. Fig. 6.3b). This impact is also reflected on the size of the branch-and-bound tree, in particular on tiny instances (see Fig. 6.3c): regarding all instance classes and applying either isolation or capacity cuts, the average reduction was roughly 85% on small instances (see Fig. 6.3d).

On the other hand, these cuts did not present a relevant outperformance when applied on instances

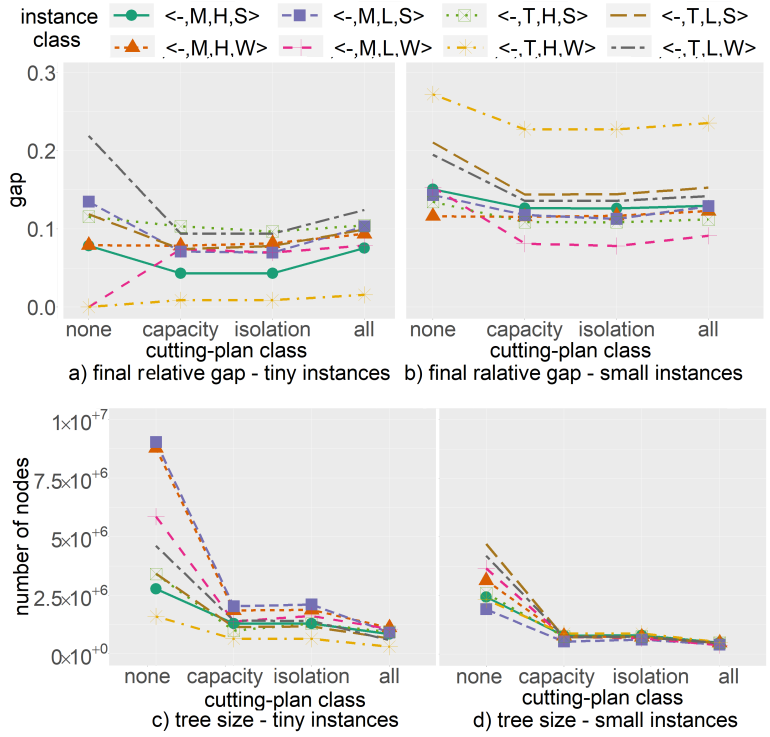


Figure 6.3 – Impact of different cutting-planes.

without strict capacity, latency, and isolation constraints: see the results for $< -, M, H, W >$ instances on both instance sizes in Fig. 6.3a and Fig. 6.3b for example. Also, due to the time required by the separation routine within the Branch-and-Cut framework, jointly applying all proposed cutting-planes increased the final gap. Regarding all instance classes and sizes, the average time spent within the separation routine was 10 times greater than when only one cut class was applied. This issue might be overcome by increasing the number of threads on the parallel searches for violated inequalities.

6.5.3 Row-generation framework

Fig. 6.4 depicts the cumulative distribution function (CDF) from the tests applying the proposed RMRG framework. The search of violated cuts from the set L of lazy constraints was done as previously discussed and added using 8 parallel threads. While *BB* refers to the formulation (5.1)-(7.9) solved by the solver's Branch-and-Bound, *CBC+* and *RMRG+* are respectively CBC and RMRG approaches along with SB, LB, SP, and MC inequalities (6.1)-(**??**) and both isolation and capacity-cut classes. Finally, we generate 40 medium-size instances varying the graph density from 0.10 to 1.00

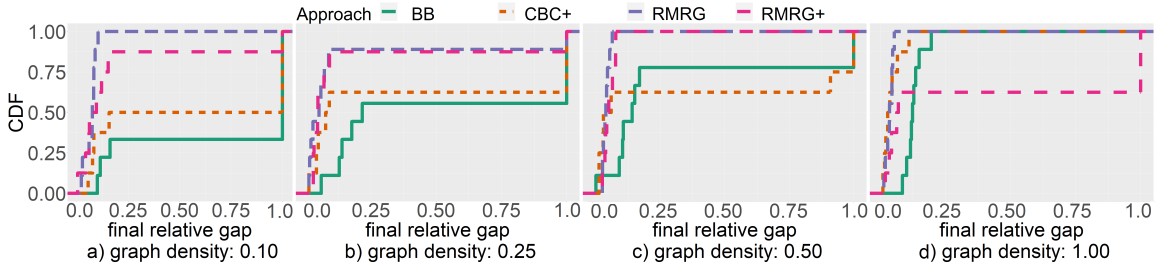


Figure 6.4 – Impact of the Row-Generation framework on medium-size instances.

(see Table 6.2). Instances with no feasible solution found within the time limit (i.e., 3600 seconds) are represented with a final gap equal to 100%.

We observe that RMRG outperformed BB in all graph densities. Indeed, while the RMRG model had a final gap smaller than 10% in more than 90% of all instances, RMRG+ has its performance decreased only on instances with complete mesh graphs (see Fig. 6.4d); these approaches had a similar efficiency on instances with graphs with density equals to 0.25 and 0.50 (see Fig. 6.4b and Fig. 6.4c, respectively). This behavior is expected since the separation routines within these approaches search for violated link-related constraints and, hence, the time to find them is directly proportional to the number of arcs in the graph. Moreover, the graph density has also an important impact on BB and CBC+ approaches. Since it is more difficult (resp. easier) to find a solution feasible for both mapping and routing sub-problems in sparse (resp. mesh) graphs that respect all capacity and latency constraints in the NSDP, BB had less (resp. more) than 35% (resp. 90%) of instances solved with a final gap smaller than 25% (see Fig. 6.4a; resp. Fig. 6.4d).

6.6 Summary

In this chapter, we presented and discussed the Network Slice Design Problem in 5G systems, and proposed a MILP formulation and exact algorithms to solve it. Numerical experiments showed the efficiency of each approach on different instance classes. For instance, the proposed symmetry-breaking and lower bound-based constraints led to an important decrease in the final gap: from 30% to 1% in some instances. Also, our Branch-and-Cut algorithm could reduce the size of the Branch-and-Bound tree by 85% and outperformed the solver's Branch-and-Bound algorithm in all tests. Finally, our Row-Generation framework outperformed the Branch-and-Bound approach in all tests, especially in those with sparse graphs.

Chapter 7

A math-heuristic for the NSDP

Contenu

The overall objective of this chapter is to go beyond the work presented in the previous chapter. To this purpose, we propose an open-access framework based on a math-heuristic for the Network Slice Design Problem present numerical results to assess the efficiency of our approach[1]. The overall idea of the proposed math-heuristic relies on decomposing the NSDP into a few sub-problems and sequentially solve them. These sub-problems are related to the following decisions: split selection, NFS-NF packing, NF-node embedding, and traffic routing. All notations follow those presented in Table 5.1.

1. The content of this chapter was published in the following paper: W. d. S. Coelho, A. Benhamiche, N. Perrot and S. Secci. "A Math-Heuristic for Network Slice Design", the 33nd International Teletraffic Congress (ITC 33), Avignon, France, 2021.

Algorithm 3: Math-heuristic for the NSDP

input : An NSDP instance $\mathbb{I}(G, S, F, N, C)$.
output: A solution to $\mathbb{I}$.

1 BestSolution, CurrentSolution $\leftarrow \emptyset$
2 **while** *a feasible solution to* $\mathbb{I}$ *is not found* **do**
3 chooseCUs()
4 **foreach** $s \in S$ **do**
5 **foreach** $k \in K(s)$ **do**
6 getPaths() ;
7 choosePaths() ;
8 selectSplit(s, CurrentSolution)
9 **while** *a feasible embedding is not found* ***or*** *maximal number of tries is reached* **do**
10 **if** $N \leftarrow$ *packNFSs() ;*
11 *is not feasible* **then** stop and go to step 3;
12 **else if** *embedNFs() fails and maximal number of tries is reached* **then** stop and go to step 3;
13 **if** *routing() fails;*
14 **then** stop and go to step 3 ;
15 **if** *CurrentSolution is feasible and cost(CurrentSolution)* $<$ *cost(BestSolution)* **then**
16 BestSolution $\leftarrow$ CurrentSolution
17 **if** *rand()* $> \rho$ **then** try to find another solution to $\mathbb{I}$ by going to step 3;
18 **return** BestSolution

7.1 Algorithm description

Algorithm 1 presents the global framework of our approach. As input, the heuristic receives an NSDP instance composed of a directed graph G representing the physical network with the set of capacities C, a set of slice requests S, each of which with a set of traffic demands $K(s)$, a set F of NFS types, and a set N of potential host virtual functions. As output, Algorithm 1 returns a virtual network to each slice request $s \in S$ ensuring all technical constraints imposed by both physical and virtual layers.

7.1.1 Split selection

After initializing the auxiliary variables at step 1, the first decision is made by the *chooseCUs()* procedure. It first calculates the minimal number of CUs to host all functions serving slices without isolation constraints: this lower-bound α is given by Equation (7.1), which considers the ratio of the most demanded physical resource by NFSs and the most critical resource on physical nodes as well as the different traffic from data and control planes.

$$\alpha = \sum_{f \in F^c} max\{c_f^c / c_u^c : c \in C, u \in V^{ac}\} \left\lceil \sum_{s \in S} \frac{n_s b_f}{cap(f)} \right\rceil$$

$$+ \sum_{f \in F^d} max\{c_f^c / c_u^c : c \in C, u \in V^{ac}\} \Big\lceil \sum_{k \in K(s): s \in S} \frac{\lambda_{f-1} b^k}{cap(f)} \Big\rceil \tag{7.1}$$

Then, the procedure builds a set V^h of α host CUs with the most centralized nodes. If it is not called for the first time, the procedure builds a new set with the nodes already chosen in previous iterations. The *Closeness Centrality* value $cent(u)$ of each node $u \in V^{ac}$ is given by Equation (7.2), which verifies the distance $dist(u, v)$, in terms of latency, between the node u to all other nodes in G.

$$cent(u) = \frac{1}{\sum_{v \in V: v \neq u} dist(u, v)} \qquad , \forall u \in V \tag{7.2}$$

Next, steps 3-7 are dedicated to find elementary paths to the related traffic demands of each slice request $s \in S$, and to select a split setting to the related data-plane flow. First, the *Yen()* procedure is called to find θ paths between the origin o_k and target t_k nodes of each traffic demand $k \in K(s)$. These paths are generated by Yen's algorithm [23] in order to find only paths that respect the end-to-end latency d_s while traversing as many host CUs (those found in step 3) as possible. Since finding paths for each slice request is done independently, this procedure is run in parallel, with each thread being responsible for running Yen's algorithm on a single traffic demand. Subsequently, a path is chosen for each traffic demand by procedure *choosePaths()* in step 7, which seeks to maximize the number of host CUs visited by the selected paths. For this purpose, let x_p^k be a binary variable that takes the value 1 if path p from the set of paths $P(k)$ is chosen to carry the flow of traffic demand $k \in K(s)$; 0 otherwise. Also, let z_{uv} be an integer variable corresponding to the number of active paths passing by node $u \in V^h$ before node $v \in V^h$, with $V^h \subseteq V^{ac}$ being the set of host CUs generated in step 3. Finally, let π_{uv} be the associated cost of each pair of nodes $u, v \in V^h$. In order to break the inherent symmetry of the proposed formulation, we set $\pi_{uv} = 1 + 10^{-4}$ and $\pi_{vu} = 1 - 10^{-4}$ for any pair of nodes $u, v \in V^h$. Then, the *choosePaths()* procedure solves the following Integer Linear Program:

$$max \sum_{u,v \in V^h | u \neq v} \pi_{uv} z_{uv} \tag{7.3}$$

$$\sum_{p \in P(k)} x_p^k = 1 \qquad , \forall k \in K(s) : s \in S \tag{7.4}$$

$$\sum_{k \in K(s)} \sum_{p \in P(k)} \lambda_{uv}^p x_p^k = z_{uv} \qquad , \forall u, v \in V^h | u \neq v \tag{7.5}$$

$$x_p^k \in \{0, 1\} \qquad , \forall k \in K(s) | s \in S, \forall p \in P(k) \tag{7.6}$$

$$z_{uv} \in \mathbb{N}_0 \qquad , \forall u, v \in V^h | u \neq v \tag{7.7}$$

Since the aim is to share ordered data-plane NFS chains among as many traffic demands as possible, the objective function (7.3) consists in maximizing the number of chosen paths that have similar structures. For this end, Equation (7.5) calculates how many activated paths pass by node $u \in V^h$ before node $v \in V^h$, where λ_{uv}^p is an auxiliary binary parameter that holds 1 if node $u \in V^h$ comes before node $v \in V^h$ in path p, and 0 otherwise. While Equation (7.4) ensures that there is exactly one path selected to each traffic demand k, (7.6) and (7.7) are the variable domain constraints. The proposed formulation is then solved by a distributed parallel branch-and-bound algorithm [127] using multiple threads to solve each singular branch-and-bound tree's node.

After, a split setting is selected for each slice in step 8. For this purpose, the *selectSplit()* procedure verifies if there is at least one common host CU visited by the chosen path of each traffic demand $k \in K(s)$ of a given slice $s \in S$. If that is the case, a split setting is randomly chosen for a random sub-set $S' \subseteq S$ of slices requests as follows. For each slice request $s \in S'$, an NFS $f \in F^d$ is randomly chosen to be the first centralized NFS in the data-plane chain. Then, all NFSs before the chosen NFSs f in the ordered set F^d are set to be distributed. The remain sub-set $S \backslash S'$ of slice requests are set to have all their data-plane NFSs centralized. On the other hand, if there is no common host CU visited by the chosen path of each traffic demand $k \in K(s)$ of a given slice $s \in S$, all related data-plane NFSs are forced to be distributed, that is, they must be installed in each DU node $u \in O(s)$. Let us recall that control-plane NFSs cannot be distributed and hence are always installed in aggregation/core nodes. Since selecting a split for each slice request is done independently, the *selectSplit()* is run in a parallel way, with each distributed thread being responsible for a single slice. As output, sets $F^{dist}(s)$ and $F^{cent}(s)$ are generated with the distributed and centralized network function service copies, respectively.

7.1.2 Network function service packing

The *packNFSs()* procedure (see Algorithm 2) generates copies of network functions with different NFS types in order to process the data from all slices. This decision is made by translating the related NSDP sub-problem into a Vertex Coloring Problem [93]. To this end, let $distGraph = (V^d, E^d)$ be the conflict graph associated with the set of distributed NFSs as follows. A node u in V^d is associated with every tuple $(s, f, u) : s \in S, f \in F^{dist}(s), u \in O(s)$ and there exists an edge in $(v, v') \in E_d$ between any two nodes $v = (s, f, u)$ and $v' = (s', f', u')$ from V^d if $(q_{ff'}^{ss'} q_{f'f}^{s's} = 0) \vee (u \neq u')$ holds.

Algorithm 4: packNFSs()

input : An NSDP instance $\mathbb{I}$ and a partial solution *CurrentSolution* to $\mathbb{I}$.
output: A set N of network functions with the related hosted NFSs.

```
1  distGraph ← getDistributedConflictGraph()
2  distCliqueSize ← getClique(distGraph)
3  colorGraph(distGraph, distCliqueSize)
4  N ← buildNFs(distGraph)
5  centGraph ← getCentralizedConflictGraph()
6  centCliqueSize ← getClique(centGraph)
7  while a feasible set of NFs is found or the number of tries is reached do
8      colorGraph(centGraph, centCliqueSize)
9      N' ← buildNFs(centGraph)
10     foreach centralized NF n ∈ N', capacityc ∈ C do
11         if Σ_{f∈n} c_f^c > max{c_u^c : c ∈ C, u ∈ V^h} then
12             if number of tries is not reached then
13                 stop and go to step 8
14             else
15                 stop and return no solution
16 return N ∪ N'
```

Hence, an edge in *distGraph* exists in order to forbid two NFSs to be packed together in the same function n while violating the isolation constraints imposed by the NSs, or slices s and t do not share any access node $u \in V^{du}$. Once the conflict graph is generated in step 1 in Algorithm 2, its maximum clique size is calculated with *getCliquesSize()* on step 2 by applying the Grimmett-McDiarmid's greedy algorithm [128]: in each iteration, a random vertex is chosen and added to the current clique if and only if it is a common neighbor.

In order to find a clique with maximum size, the greedy algorithm is run several times in a distributed parallel way (i.e., across multiple threads) within the *getCliquesSize()* procedure. The best value is then taken into consideration as lower-bounds to the related vertex coloring problem. This translated sub-problem is then solved in step 3 by the *colorGraph()* procedure, which runs the randomized sequential coloring algorithm presented by Syslo [129]. It is also run several times in a distributed parallel way and returns the best coloring (i.e., with the minimal chromatic number). Regarding the related clique size calculated in step 2, the *colorGraph()* procedure stops any time an optimal coloring is found (i.e., the clique size equals to the chromatic number) or a maximal number of tries is reached. Hence, each NFSs represented by a vertex with the same color is packed in the same network function by the *buildNFs()* procedure in step 4.

Steps 7-15 in Algorithm 2 repeat the previous ones in order to pack the centralized NFS set, which also includes control-plane network function services. For this purpose, let $centGraph = (V^c, E^c)$ be

the associated conflict graph as follows. For each centralized NFS $f \in F^{cent}(s)|s \in S$, we associate a node v in V^c with every tuple (s, f). Also, let $w_f^s \in \mathbb{R}_+$ be the ratio between the quantity of traffic from slice s and processed by NFS f, over its capacity $cap(f)$ as calculated by Equation (7.8).

$$w_f^s = \begin{cases} \dfrac{n_s b_f}{cap(f)} & \text{if } f \in F^c; \\ \dfrac{\sum\limits_{k \in K(s)} \lambda_{f-1} b_k}{cap(f)} & \text{if } f \in F^d. \end{cases} \quad ,\forall s \in S, \forall f \in F^{cent}(s) \tag{7.8}$$

Moreover, there exists an edge in $(v, v') \in E_c$ between any two nodes $v = (s, f)$ and $v' = (s', g')$ from V^d if $(max\{\dfrac{c_f^c w_f^s + c_{f'}^c w_{f'}^{s'}}{c_u^c} : c \in \ C, u \ \in \ V^h\} > 1) \vee (q_{ff'}^{ss'} q_{f'f}^{s's} + q^{ss'} q^{s's} < 2)$ holds. Hence, an edge in *centGraph* exists in order to forbid two NFSs to be packed together in the same function n while violating capacity and isolation constraints. Since the conflict graph is built regarding only each pair of NFSs, the coloring solution calculated in step 8 must be checked: if the sum of physical capacity required by all NFSs hosted by a given NF $n \in N'$ is greater than the maximal amount that any host CU can provide, another coloring for the related conflict graph must be provided if the maximal number of tries is not reached; otherwise, *packNFSs()* procedure will be stopped and no solution will be generated.

7.1.3 Network function embedding

In step 12 from Algorithm 1, each network function $n \in N$ generated in the previous step is embedded into a physical node. A copy of each n hosting a distributed NFS $f \in F^{dist}(s)$ from any slice $s \in S$ is generated and embedded into every associated DU node $u \in O(s)$. For network functions hosting centralized data-plane NFSs, a host node is chosen as follows. Let $S(n)$ be the set of slices from S and served by NF $n \in N$, and $V(n)$ the set of host CUs among those visited by all paths $p \in P(k) : s \in S(n), k \in K(s)$ as chosen in step 7 of Algorithm 1. Then, for each NF n hosting centralized network functions services, a host physical node is randomly chosen among those in $V(n)$. This procedure is repeated until an embedding that respects capacity constraints on physical nodes is found or the maximal number of tries is reached.

7.1.4 Traffic routing

The last sub-problem solved within Algorithm 1 is related to finding a path for each pair of physical nodes that host NF copies that must be connected. Algorithm **??** depicts our approach to

Algorithm 5: routing()

input : An NSDP instance $\mathbb{I}(G, S, F, N, C)$ and a Current Solution to the problem.
output: A path to each pair $u, v \in V$ hosting NFs that must be connected.

1 **foreach** $u, v \in hostPairs$ **do**
2 *paths(u,v)* $\leftarrow$ *getPaths();* `/* By Yen's algorithm */`
3 eliminate all paths in paths(u,v) that does not support the expected traffic volume between u and v
4 **if** $paths(u, v) = \emptyset$ **then** stop and **return** $\emptyset$;
5 **while** *the maximal number of tries is not reached* **do**
6 randomly choose a path to each pair of nodes in hostPairs
7 **if** *all paths respect link capacity and end-to-end latency constraints* **then**
8 **return** selected paths
9 **else**
10 stop and go to step 6
11 **return** $\emptyset$

generate a solution to this sub-problem. First, let $hostPairs$ be the set of such pair of nodes. Then, a set $paths(u, v)$ of paths to each pair of nodes in $hostPairs$ is generated. For this purpose, Yen's algorithm [23] is used to find θ shortest paths between each node pair $(u, v) \in hostPairs$ that respects the maximal latency imposed by related NF copies. Then, all paths that do not support the expected volume between the related NFs are deleted. Since finding paths for each flow is done independently, this procedure is run in a distributed parallel way: each thread is responsible for running Yen's algorithm for a given flow and verifying the capacity constraints. Finally, a path from $path(u, v)$ is randomly chosen for each $u, v \in hostPairs$. If the combined traffic volume of the selected paths does not respect the capacity of the related physical links, another selection of paths is made. The procedure returns no path if the number of tries of path selection is reached or no feasible path is generated in steps 1-4.

7.1.5 Final solution

If a feasible solution is generated, its cost is then calculated in step 15 of Algorithm 1 as follows:

$$cost(Solution) = \sum_{f \in F} \sum_{n \in N} \sum_{u \in V} \sum_{c \in C} \mu_u^c y_{nu}^f \tag{7.9}$$

Where $y_{nu}^f \in \mathbb{Z}_0$ and μ_u^c are the total number of NFSs f packed into NF n and installed on node u, and the related cost per unit of resource usage $c \in C$, respectively. It is worthwhile mentioning that the first cost of *BestSolution* is set to $+\infty$ in step 1. Then, step 17 tests if a better solution should be

found, where $rand()$ uniformly generates a random number between 0 and 1 and $\rho(t) = 1 - \phi/t$ is a function depending on the time t (in seconds) that passed until such verification and a parameterizable value ϕ. For instance, setting ϕ to 60, the probability $P(rand() \geq \rho)$ that $rand()$ is greater than ρ is equal to 100% if t is equal to 60 seconds or less, and less than 50% (resp. 10%) if t is equal to 120 (resp. 600) seconds. If the test returns true, then another solution is generated. Otherwise, the best solution found theretofore is returned as output and the procedure stops.

7.1.6 Algorithm's time complexity

Table 7.1 – Time Complexity

Main Procedures	Asymptotic Complexity
chooseCU()	$O(F + V)$
getPaths()	$O(KV^4)$
choosePaths()	Solver's black box's complexity
selectSplit()	$O(SF^d)$
packNFSs()	$O(SF + V^3 + N)$, V from conflict graph
embedNFs()	$O(N)$
colorGraph()	$O(V^2)$, V from conflict graph
routing()	$O(V^4)$
Auxiliary Procedures	**Asymptotic Complexity**
getDistributedConflictGraph()	$O(SF^d)$
getCentralizedConflictGraph()	$O(SF)$
getCliqueSize()	$O(V^4)$, V from conflict graph
BuildNFs()	$O(N)$

Table 7.1 summarizes the time complexity of each procedure within Algorithm 1, where $K = \sum_{s \in S} |K(s)|$. All other notations follow those presented in Table 5.1. Let us recall that, besides chooseCU() and embedNFs(), all procedures are able to be run in a distributed parallel way.

7.2 Numerical experiments

Let us first detail the simulation settings. We propose different instance sizes (see Table 7.2), in which we set the processing capacity $cap(f)$ of each NFS in F^d to between 50% and 100% of the

Table 7.2 – Instance Sizes

Instance size	$\|V\|$	Graph Density*	$\|S\|$	$\|K\|$	$\|F^d\|$	$\|F^c\|$
Tiny (T)	10	0.15	2	1	2	2
Small (S)	15	0.10	2	2	4	2
Medium-Small (SM)	20	0.15	4	3	4	3
Medium (M)	25	0.15	4	8	6	4
Medium-Big (MB)	30	0.20	4	8	6	6
Big (B)	35	0.20	8	8	8	6
Extra-Big (EB)	40	0.25	8	8	8	8

* Ratio between exiting and theoretically possible number of arcs.

Table 7.3 – Instance Classes

Latency	Description
Low (L)	The maximum latency d_{fg} between two NFSs and the end-to-end latency d_s imposed by each slice request $s \in S$ is set respectively to between 50% and 150% and to between 250% and 500% of the average latency on the physical links.
High (H)	The maximum latency d_{fg} between two NFSs and the end-to-end latency d_s of each slice $s \in S$ is set respectively to between 200% and 400% and to between 300% and 1000% of the average latency on the physical links.
Capacity	**Description**
Tight (T)	The available bandwidth b_a on the physical links have between 50% and 100% of the average volume (without compression) generated by the slices. In addition, each physical node $u \in V \backslash V^{ap}$ has enough capacity to host between 1 and 3 copies of each NFS type; application nodes has no available capacity.
Moderate (M)	The available bandwidth b_a on the physical links have between 200% and 300% of the average volume (without compression) generated by the slices. In addition, each physical node $u \in V \backslash V^{ap}$ has enough capacity to host between 5 and 8 copies of each NFS type; application nodes has no available capacity.
Isolation	**Description**
Weak (W)	10% of isolation parameters q^{st} and q^{st}_{fg} are set to 0; they are randomly chosen.
Strong (S)	75% of isolation parameters q^{st} and q^{st}_{fg} are set to 0; they are randomly chosen.

average volume generated by the traffic demands. For NFSs of F^c, this value was set to between 50% and 100% of the volume related to the total number n_s of expected UEs connected to the slice. Also, the total amount b_{fg} of traffic between two functions from $F(s) \cup G(s)$ was set to 1 Kbps per UE. As shown in Table 7.3, different instance classes are also proposed, which are related to the ratio between the resource required by the slices and those available on the physical network, and also between the latency on the physical links and the threshold imposed by slices and pairs of NFSs. The complete reference of each generated instance is given by joining the acronyms of each size/class name from tables 7.2 and 7.3. For example, $< S, L, M, S >$ refers to a *small* instance with *low* latency threshold, *moderate* capacity requirements, and *strong* isolation constraints. We implemented our model in a Julia-JuMP environment using ILO CPLEX 12.10 as the linear solver. Our tests were run on a Linux server with an Intel Xeon E5-2650 CPU. Also, we provided 12 threads for each distributed parallel procedure. Finally, for each instance size/class, 30 different instances were randomly generated as previously discussed. The data-set and the source code are available on [130].

7.2.1 Quantitative analyses

We first analyze the efficiency of the proposed Math-Heuristic. For this purpose, we compare it with the mixed-integer linear programming formulation (5.1)-(5.19) with the proposed objective function (6.29), hereafter referred to as to *MILP*, introduced in the previous chapters. Concerning

step 17 of Algorithm 1, we set ρ to $1 - 60/seconds$ for tiny and small instances and $1 - 600/seconds$ for all other sizes in our simulations.

Fig. 7.1 shows different results on tiny and small instances. First, as seen in Fig. 7.1a and Fig. 7.1f, approximately 75% (resp. 70%) of all tiny (resp. small) instances were solved in less than 130 (resp. 220) seconds. We also observe that tiny (resp. small) instances with moderate (resp. tight) capacity constraints were solved faster in general. For instance, the average runtime was roughly 98 and 118 (resp. 85 and 120) seconds on $< T, M, L, W >$ and $< T, T, L, W >$ (resp. $< S, T, L, W >$ and $< S, M, L, W >$) instance classes, respectively. While there is no strong impact on the number of feasible solutions found by the Math-heuristic on small-size instances (see Fig. 7.1g), this behavior led to an increase in the number of solutions for tiny-size instances (see Fig. 7.1b). In fact, 75% of $< T, M, L, W >$ and $< T, M, L, S >$ (resp. $< T, T, H, W >$ and $< T, T, L, W >$) instance classes had more (resp. less) than 1700 (resp. 900) feasible solutions found within 180 seconds of execution time. Considering all instance classes, the average time needed to find a feasible solution for tiny and small instances was approximately 1 and 3 seconds, respectively. Moreover, as seen in Fig. 7.1c and Fig. 7.1h, the best solution (not necessarily the optimal one) was found in less than 300 rounds for more than 50% of all tiny and small instances. Due to greater feasible solution space, more iterations were needed for instances with moderate capacity constraints. In fact, the best solution could be found only after 800 (resp. 1000) rounds for some $< T, M, L, S >$ (resp. $< S, M, L, W >$) instances.

Fig. 7.1d and Fig. 7.1i depict the final gap related to the best solution found by the proposed Math-heuristic and the optimal value obtained by MILP. First, we observe that more than 80% (resp 75%) of tiny-size (resp. small-size) instances had a gap smaller than 2% (resp. 4%). Also, we do not observe any strong impact on the final gap related to different instance classes. In fact, 100% of all instances solved by the Math-Heuristic had a final gap less or equal to 10%. However, the average gap of each feasible solution for tiny-size instances was better when strong isolation and strict latency constraints were applied. As seen in Fig. 7.1e and Fig. 7.1j, the average gap on $< T, T, L, S >$ and $< T, M, H, W >$ (resp. $< S, M, L, S >$ and $< S, T, H, W >$) instances were respectively 2% and 5% (resp. 5% and 8%).

Table 7.4 shows the total execution time and final gap on medium-small, medium, medium big, big, and extra big instances. In each simulation, an instance class (i.e., a combination of capacity, latency, and isolation constraints; see Table 7.3) was randomly chosen. While the third and fourth

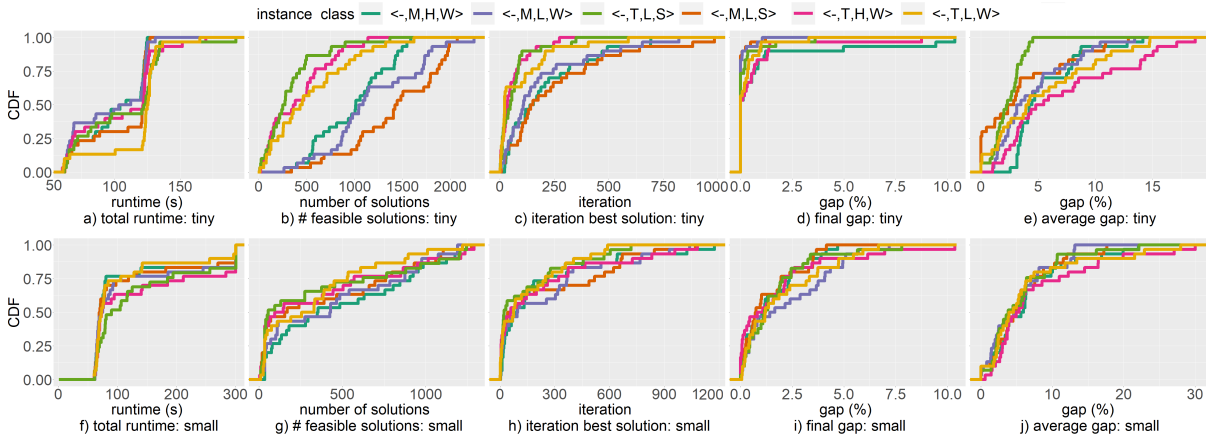


Figure 7.1 – Quantitative analyses: tiny and small instances.

columns of Table 7.4 are related to MILP, the two last columns depict the values when the proposed Math-Heuristic was applied. In both cases, the final gap is related to the best solution and the lower bound obtained from the linear relaxation of MILP. First, we observe that the Math-Heuristic was faster than MILP in all instance sizes. While MILP reached the time limit in almost all instance sizes, our Math-Heuristic needed less than 20 minutes to find a good solution. As seen in Table 7.4, the average gap could be reduced from 36.3% to 4.3% on medium-big instances. Moreover, while MILP could not find any feasible integer solution for big and extra big instances within 1 hour, the average runtime and final gap were respectively 997 seconds and 8.2% (resp. 1245 seconds and 11.5%) when Math-Heuristic was applied on big (resp. extra big) instances.

7.2.2 Qualitative analyses

To better understand the impact of our math-heuristic on the physical network, we now analyze different parameters related to both physical nodes and physical links, as well as the end-to-end (e2e) latency on the data-plane flow. Fig. 7.2 shows the impact of each approach on the aforementioned

Table 7.4 – Quantitative analyses: from medium small to extra big instances

Instance Size	MILP		Math-Heuristic	
	Runtime (s)	Gap (%)	Runtime (s)	Gap (%)
Medium-Small	2165 ± 364	0	**732 ± 87**	3.2 ± 0.5
Medium	3600*	5.7 ± 2.1	**803± 122**	4.5±0.7
Medium-Big	3600*	36.3 ± 4.8	**894 ± 109**	**4.3±1.2**
Big	3600*	**	997 ± 84	**8.2±3.6**
Extra-Big	3600*	**	**1245 ± 241**	11.5±2.4

* Time limit reached. ** No feasible solution was found.

network parameters. Compared to MILP, more physical links are used to carry the expected slice flows in the final solution found by the proposed Math-Heuristic. For instance, the average link usage ratio was respectively 62% and 76% (resp. 53% and 58%) on $< T, M, H, S >$ and $< S, M, H, S >$ when Math-heuristic (resp. MILP) was applied, (see Fig. 7.2a and Fig. 7.2f). However, the average load on active links remains the same on tiny instances (see Fig. 7.2b) and on small instances without strict capacity constraints (see Fig. 7.2g). Also, as seen in Fig. 7.2g, the Math-heuristic could reduce the average load on active links on small instances with strict capacity constraints, especially those with strong isolation restrictions: compared to MILP, we observe a reduction from 58% (resp. 55%) to 40% (resp. 38%) on $< S, T, L, S >$ (res. $< S, T, H, S >$) instances. Regarding all instance classes, the average reduction was approximately 16% on small instances. Finally, we observe an adverse effect on the e2e latency only on small instances without strict capacity constraints applying the proposed Math-Heuristic, especially on instances without strict latency constraints. Even though all latency constraints are respected (i.e., e2e latency and between each pair of NFSs) by both approaches, the average e2e data-plane latency was respectively 0.5 ms and 2.55 ms (resp. 0.38 ms and 1.89 ms) on $< T, M, H, S >$ and $< S, M, H, W >$ when Math-Heuristic (resp. MILP) was applied; regarding all instance classes, the average increase on the e2e latency was equal to 14% and 25% on tiny and small instances, respectively (see Fig. 7.2c and Fig. 7.2h).

We also observe similar behavior on physical nodes. The proposed Math-Heuristic led to more nodes hosting at least one network function than the MILP formulation. As seen in Fig.7.2i, the number of host nodes considerably increased when the Math-heuristic was applied on small instances, especially those with strong isolation constraints. For instance, the ratio of host nodes increased from 17% (resp. 17%) with MILP to 36% (resp. 27%) with Math-Heuristic on $< S, T, L, S >$ (resp. $< S, M, L, S >$) instances; regarding all instance classes, the average ratio increased by roughly 2% and 47% on tiny and small instances, respectively (see Fig.7.2d and Fig.7.2i). This behavior, however, led to an important decreasing in the average node load, especially on small-size instances with strict capacity and isolation constraints. As seen in Fig. 7.2j, the Math-heuristic could reduced the average load on host nodes from 41% (resp. 30%) to 18% on $< S, T, L, S >$ (res. $< S, T, H, S >$) instances. Regarding all instance classes, the average reduction was approximately 35% on small instances (see Fig. 7.2j); we did not observe an important impact on the average load of physical nodes on tiny instances (see Fig. 7.2e).

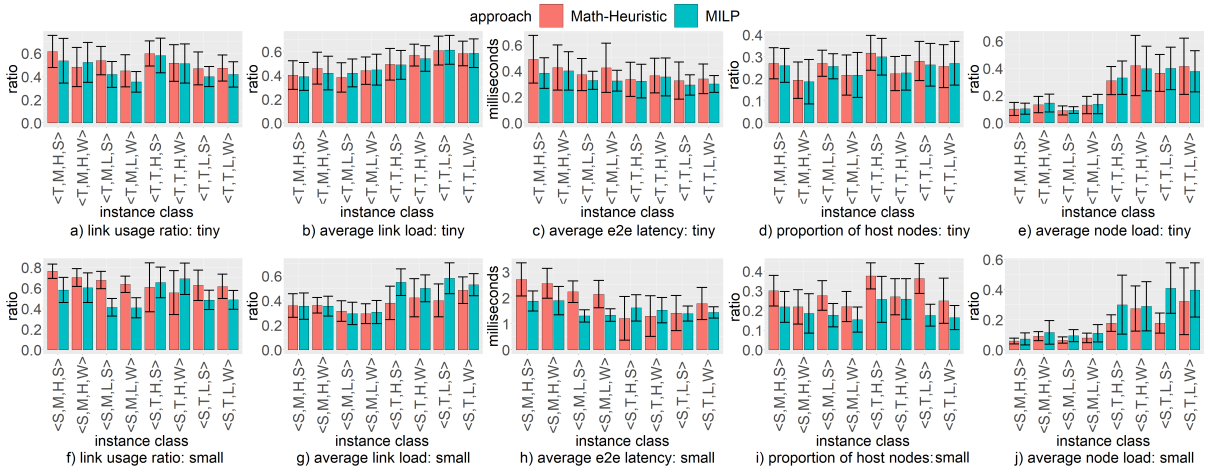


Figure 7.2 – Qualitative analyses: tiny and small instances.

7.3 Summary

In this chapter, we presented and discussed the Network Slice Design Problem in 5G systems, proposing an open-access framework based on a Math-heuristic to address the underlying optimization problem. The overall idea of the proposed approach relies on decomposing the NSDP into several sub-problems and sequentially solve them while encompassing control-plane and data-plane separation and novel mapping and decomposition dimensions influencing the placement and interconnection of slices. Numerical experiments showed the efficiency of our approach on different instance classes, which could attain near-optimal solutions in a competitive runtime. Comparing it to a mixed-integer linear programming formulation, the proposed Math-Heuristic could reduce the average runtime and the final gap by up to 78% and 90%, respectively. Moreover, our approach could reduce the congestion on the physical network, better balancing the data flow while considering all technical constraints. For instance, the average load on physical links and physical nodes could be reduced by 16% and 35%, respectively.

On a practical note, as our Math-Heuristic could reduce the average load on physical nodes and physical links, a tough but interesting extension is to use it within an online algorithm. Our approach might potentially increase the slice acceptance ratio, that is the ratio between the number of embedded slices and the number of requests, since the solution proposed by the Math-Heuristic distributes the data flow among several nodes and links, leading to a decreasing in the network congestion. More tests

must therefore be carried out to assess the effects of the proposed Math-heuristic on such scenarios and before conclusions can be drawn. Also, let us recall that, as the proposed approach is a Math-Heuristic, the optimality of the solutions found by our algorithm cannot be ensured. However, as seen in the presented numerical results, the efficiency of our approach offsets this aspect and led to finding solutions with a small average gap on relatively short execution time, even on big instances. Thus, it would be interesting and most probably very powerful to use it as a primal heuristic to boost the efficiency of an exact algorithm.

Chapter 8

Network slice design with dedicated network functions

Contenu

In this chapter, we present another variant of the NSDP, where the Hard Isolation policy is always applied, that is, all network functions are slice-dedicated (see Chapter 3 for more information). Thanks to this hypothesis, the complexity of the studied problem can be decreased, allowing us to propose different heuristic and mathematical approaches to solve the related optimization problem. However, the intra-slice anti-affinity constraints remain, which represent the possibility of packing two different NFSs into the same network function serving a given network slice. The motivation for keeping the NFS-NF mapping comes from decreasing the operational complexity related to the deployment of each NS. In other words, installing few network functions instead of several network function services allows operators to aggregate into the same entity all NFSs that share the same communication protocol or other types of affinity. Hence, the set-up step of deploying networks slices might potentially be easier and faster, reducing, therefore, operating expenses costs [131]. We refer this variant as to *Network Slice Design with Dedicated Network Functions* (NSDP-DNF).

In what follows, we present different strategies in order to efficiently solve the optimization problems related to the NSDP-DNF[1]. First, we propose a compact formulation and an extended one. To solve the former, we proposed a Relax-and-Fix heuristic that relies on repetitively solving the proposed related ILP with only a few integer variables and fixing or relaxing most of the remaining integer and binary ones. On the other hand, to address the exponential number of variables in the extended formulation, a column generation-based framework is then proposed.

8.1 A compact formulation for the NSDP-DNF

In this section, we present a integer linear programming formulation for the NSDP. The following formulation is based on those proposed on our previous Chapter 5, and presents some modifications to better address the variant of the NSDP-DNF presented in this chapter. The problem variables are therefore defined as :

- $z_f^s \in \{0,1\}$ that takes value 1 if NFS f is centralized, 0 otherwise,

- $x_{nu}^{sf} \in \{0,1\}$ that takes value 1 if NFS f serving slice s is packed into NF n, installed on node u, 0 otherwise,

- $w_{nu}^{sf} \in \mathbb{Z}$ the total number of NFSs f packed into NF n, installed on node u, and serving slice s

- $y_n \in \{0,1\}$ that takes value 1 if NF n is built; 0 otherwise,

- $\gamma_{fg}^{ka} \in \{0,1\}$ that takes value 1 if arc a is used to route the traffic between NFS f and NFS g for demand k, 0 otherwise.

- $r_a \in \mathbb{R}_+$ the total volume traversing arc a.

The constraints are categorized into several blocks and defined as follows :

Split Selection: The *split selection constraints* are to impose that if NFS $f \in F^d$ is centralized, then the remaining NFSs of the ordered set F^d should also be centralized, thus defining a splitting policy

1. The content of this chapter was subbimited to Networks Journal in November 2021 as: W. d. S. Coelho, A. Benhamiche, N. Perrot and S. Secci, "Relax-and-Fix and Column Generations Algorithms for the Network Slice Design Problem".

attached to a slice :

$$z_f^s \leq z_{f+1}^s, \qquad \forall s \in S, \forall f \in F^d \backslash \{f_{|F^d|}\}. \tag{8.1}$$

Network Function Services Dimensioning and Packing: Constraints (8.2) are to define variables w directly as a function of x variables while forcing to install enough NFS to process all the traffic: they express the number of copies of distributed and centralized NFSs needed for each NS request $s \in S$. Inequalities (8.3) are the anti-affinity constraints on NFs on the virtual layer. They also ensure that NF n is assigned to at most one physical node of V, while forcing the variable y to be equal to 1 if there is at least one NFSs hosted by the related network function $n \in N$.

$$cap(f) w_{nu}^{sf} \geq \begin{cases} \sum\limits_{k \in K(s) | u = o_k} \lambda_{f-1} b^k x_{nu}^{sf} & \text{if } f \in F^d, u \in V^{du} \\ n_s b_f x_{nu}^{sf} & \text{if } f \in F^c; \\ \sum\limits_{k \in K(s)} \lambda_{f-1} b_k x_{nu}^{sf} & \text{if } f \in F^d, u \in V^{ac}. \end{cases} \qquad \forall s \in S, \forall f \in F, \forall n \in N, \forall u \in V \tag{8.2}$$

$$x_{nu}^{sf} + x_{nv}^{tg} \leq \begin{cases} 1 + q_{fg}^s & \text{if } s = t, u = v, \\ y_n & \text{otherwise .} \end{cases} \qquad \forall s, t \in S, \forall f, g \in F, \forall n \in N, \forall u, v \in V \tag{8.3}$$

Network Function Services Placement: For each slice request s, equalities (8.4) ensure that a distributed NFS f should be installed on every origin node of $K(s)$; whereas equalities (8.5) ensure that each centralized NFS f serving s should be installed in a node of V^{ac}.

$$\sum_{n \in N} x_{nu}^{sf} = \begin{cases} 1 - z_f^s & , \text{ if } f \in F^d, u \in O(s); \\ 0 & , \text{ otherwise.} \end{cases} \qquad , s \in S, \forall f \in F, u \in V^{du} \tag{8.4}$$

$$\sum_{n \in N} \sum_{u \in V \backslash V^{du}} x_{nu}^{sf} = \begin{cases} z_f^s & , \text{ if } f \in F^d; \\ \alpha_f^s & , \text{ otherwise.} \end{cases} \qquad s \in S, \forall f \in F \tag{8.5}$$

Traffic routing: The constraints (8.6) are the flow conservation constraints, for each slice request $s \in S$, each demand $k \in K(s)$ and each pair of NFSs in F. They allow to associate a path in G for each traffic demand k between its origin node o_k and the first NFS $f = 1$ from the ordered set F^d and serving k, and between the last NFS $f = |F^d|$ from F^d and the destination node t_k of k.

$$\sum_{a\in\delta^+(u)} \gamma^{ka}_{fg} - \sum_{a\in\delta^-(u)} \gamma^{ka}_{fg} = \begin{cases} z^s_f - 1 & , \text{ if } (f,g)\in G(s), f\in F^c, u = o_k, \\ 1 - z^s_f & , \text{ if }, u = o_k, (f,g)\in G(s), f\in F^d \\ & \text{ or if } u = o_k, f = f_{|F^d|}, g = f_0; \\ -\sum\limits_{n\in N} x^{sg}_{nu} & , \text{ if } u\in V\backslash V^{du}, f = f_0, g = f_1 \\ z^s_g & , \text{ if } u = o_k, f = f_0, g = f_1 \\ -1 & , \text{ if } u = t_k, f = f_{|F^d|}, g = f_0 \\ \sum\limits_{n\in N} x^{sf}_{nu} & , \text{ if } u\in V\backslash V^{du}, f = f_{|F^d|}, g = f_0 \\ z^s_g - z^s_f & , \text{ if } u = o_k, \forall f,g\in F^d | g = f+1 \\ \sum\limits_{n\in N} x^{sf}_{nu} - \sum\limits_{m\in N} x^{sg}_{mu} & , \text{otherwise.} \end{cases}$$

$$\forall k\in K(s) : s\in S, \forall f,g\in F, \forall u\in V \tag{8.6}$$

Latency: Inequalities (8.7) ensure that each demand $k\in K(s)$ is routed along a path that respects the end-to-end latency value requested for slice $s\in S$ while inequalities (8.8) ensure that the maximum latency value between any pair of NFSs is also respected.

$$\sum_{a\in A_p} d_a(\gamma^{ka}_{f_{|F^d|}f_0} + \sum_{f\in\{f_0\}\cup F^d\backslash\{f_{|F^d|}\}} \gamma^{ka}_{ff+1}) \leq d_s \qquad , \forall k\in K(s) : s\in S \tag{8.7}$$

$$\sum_{a\in A_p} d_a\gamma^{ka}_{fg} \leq d_{fg} \qquad , \forall k\in K(s) : s\in S, \forall f,g\in F \tag{8.8}$$

Physical Capacity: Inequalities (8.9) are the capacity constraints over the arcs of A. Note that the flow using an arc $a\in A$ is composed of two types of traffic for each $s\in S$, namely the part of traffic generated by inter-NFS communication and the part of traffic generated by the demands of $K(s)$, which is submitted to the compression coefficients λ. Inequalities (8.10) express the capacity constraints in terms of NFs that can be installed on each physical node $u\in V$.

$$\sum_{s\in S}\sum_{k\in K(s)} b^k(\lambda_{f_{|F^d|}}\gamma^{ka}_{f_{|F^d|}f_0} + \sum_{f\in\{f_0\}\cup F^d\backslash\{f_{|F^d|}\}} \lambda_f\gamma^{ka}_{ff+1})]$$

$$+\sum_{s\in S} n_s(\sum_{(f,g)\in F(s)} b_{fg}\gamma^{k_s a}_{fg} + \sum_{(f,g)\in G(s)}\sum_{k\in K(s)} \frac{b_{fg}\gamma^{ka}_{fg}}{|K(s)|}) = r_a \qquad , \forall a\in A_p \tag{8.9}$$

$$\sum_{n\in N}\sum_{f\in F} c^c_f y^f_{nu} \leq c^c_u \qquad \forall u\in V, \forall c\in C \tag{8.10}$$

The NSDP-DNF is then equivalent to the following formulation:

$$min \sum_{n\in N} y_n + \sum_{s\in S}\sum_{f\in F}\sum_{u\in V}\sum_{c\in C} c^c_f\mu^c_u w^{sf}_{nu} \tag{8.11}$$

subject to

$$(8.1) - (8.10)$$

$$0 \leq r_a \leq b_a \quad \in \mathbb{R} \quad ,\forall a \in A \tag{8.12}$$

$$y_n \quad \in \{0,1\} \quad \forall n \in N \tag{8.13}$$

$$x_{nu}^{sf} \quad \in \{0,1\} \quad \forall s \in S, \forall f \in F, \forall n \in, \forall u \in V \tag{8.14}$$

$$z_f^s, \quad \in \{0,1\} \quad \forall s \in S, \forall f \in F \tag{8.15}$$

$$\gamma_{fg}^{ka} \quad \in \{0,1\} \quad \forall k \in K(s) : s \in S, \forall f, g \in F \tag{8.16}$$

$$w_{nu}^{sf} \geq 0 \quad \in \mathbb{Z} \quad \forall s \in S, \forall f \in F, \forall n \in N, \forall u \in V \tag{8.17}$$

The objective function consists in minimizing the total cost of the NFS copies installed over the nodes of G and the number of network functions to be built. When the costs of physical resources are unitary, it allows identifying the smallest number of NFS needed to design and embed all network slice requests in a common physical infrastructure.

8.2 An extended formulation for the NSDP-DNF

In this formulation, the concept of virtual network templates is based on the definition of *Network Slice Templates* (NST) and *Network Slice Blueprint* (NSB) respectively proposed by 3GPP [7] and NGMN [132]. Hence, a virtual network template contains the complete description of the structure, the configuration of contained components, and the plans/workflows for how to instantiate and control the NS instance. We extend the NST and NSB definitions to incorporate splitting, packing, and mapping decisions. For this purpose, we introduce the set $G(s)$ of all possible virtual network templates for slice request $s \in S$. Moreover, let $\bar{x}_{nu}^{gf}$ be a parameter that takes value 1 if NFS f is packed into NF n and installed on node u in virtual network template $g \in G(s)$; 0 otherwise. Also, $\bar{w}_{nu}^{gf}$ is the ratio between the quantity of traffic processed by NFS f, packed into NF n and installed on node u in virtual network template g, over its capacity $cap(f)$. Moreover, let $\bar{y}_n^g$ be a binary parameter that takes 1 (resp. 0) if NF n is (resp. is not) built within within virtual network template g, and by $\bar{r}_a^g$ the expect volume from virtual network template g on arc a. Finally, for each slice request, let μ_g^s be the cost of embedding virtual network template $g \in G(s)$ into physical network G. This cost takes into

consideration the cost of resources from host nodes and the number of built NFs, and is calculated as following:

$$\mu_g^s = \sum_{n \in N} \bar{y}_n^g + \sum_{f \in F} \sum_{u \in V} \sum_{c \in C} c_f^c \mu_u^c \bar{w}_{nu}^{gf} \qquad \forall s \in S, \forall g \in G(s) \tag{8.18}$$

We now propose a extended integer linear programming formulation, hereafter referred as to *E-ILP*, for the NSDP. The variables related to this formulation are then defined as :

- $\pi_g^s \in \{0,1\}$ that takes value 1 if virtual network $g \in G(s)$ is activated to serve slice $s \in S$, 0 otherwise.

The E-ILP is then equivalent to the following formulation:

$$min \sum_{s \in S} \sum_{g \in G(s)} \mu_g^s \pi_g^s \tag{8.19}$$

subject to

$$\sum_{g \in G(s)} \pi_g^s = 1 \qquad , \forall s \in S \qquad \psi_s \tag{8.20}$$

$$\sum_{s \in S} \sum_{g \in G(s)} \pi_g^s \bar{r}_a^g \leq b_a \qquad , \forall a \in A \qquad \phi_a \tag{8.21}$$

$$\sum_{s \in S} \sum_{g \in G(s)} \sum_{f \in F} \sum_{n \in N} c_f^c \bar{w}_{nu}^{gf} \pi_g^s \leq c_u^c \qquad , \forall u \in V, \forall c \in C \qquad \sigma_u^c \tag{8.22}$$

$$\pi_g^s \in \{0,1\} \qquad , \forall s \in S, \forall g \in G(s) \tag{8.23}$$

The objective function consists in minimizing the total cost of the NFS copies installed over the nodes of G and the number of network functions to be built. When the costs of physical resources are unitary, it allows identifying the smallest number of NFS needed to design and embed all network slice requests in a common physical infrastructure. Constraints (8.20) ensure that exactly one virtual network template is chosen for each slice request. Inequalities (8.21) express the capacity constraints in terms of NFs that can be installed on each physical arc $a \in A$, while inequalities (8.22) are the capacity constraints over the nodes of V.

8.3 Solving approaches for the NSDP-DNF

In what follows, we present different strategies in order to efficiently solve the proposed formulations related to the NSDP-DNF. First, we propose a compact formulation and an extended one. To solve the former, we proposed a Relax-and-Fix heuristic that relies on repetitively solving the proposed related ILP with only a few integer variables and fixing or relaxing most of the remaining integer and binary ones. On the other hand, to address the exponential number of variables in the extended formulation, a column generation-based framework is then proposed.

8.3.1 Relax-and-fix algorithm

Algorithm 6 presents the global framework of our approach. The overall idea of the proposed heuristic, hereinafter referred to as Relax-and-Fix, relies on repetitively solving the proposed ILP (8.1)-(8.17) with only a few integer variables and fixing or relaxing most of the remaining integer and binary variables. As input, the heuristic receives an NSDP-DNF instance composed of a directed graph G representing the physical network with the set of capacities C, a set of slice requests S, each of which with a set of traffic demands $K(s)$, a set F of NFS types, a set N of potential host virtual functions, and the pacing strategy $\rho \in \mathbb{Z}_+$. As output, it returns a virtual network to each slice request $s \in S$ ensuring all technical constraints imposed by both physical and virtual layers.

Steps 1-6 are responsible for initiating all parameters within Relax-and-Fix. First, an auxiliary set N^* is created with only one NF in it and a model $\mathbb{M}$ is created following the presented ILP (8.1)-(8.17). In order to accelerate the solving process, $\mathbb{M}$ is created with N^* instead of N, and Inequalities (8.3) are relaxed (i.e., they are not presented in $\mathbb{M}$). Also, the integrality constraint of each variable in $\mathbb{M}$ is removed. In step 4, the set S of slice request is ordered following a given strategy. For instance, S might be ordered in an increasing (resp. decreasing) order in terms of end-to-end latency d_s (resp. required capacity) or even in a random way. In this work, each ILP sub-problem within $\mathbb{M}$ is related to a slice request (and all related variables and constraints). Hence, a pace p represents how many slices should be embedded in each iteration. In other words, in each iteration, we restore integrality constraints on all variables related to p slices requests in S. Then, in step 5, p is set to the initial pacing strategy ρ. Finally, an auxiliary set S^* is created in step 6 with no element. This set will represent the embedded slices after each pacing within Relax-and-Fix, that is, the slice request with

Algorithm 6: Relax-and-Fix

input : An NSDP-DNF instance $\mathbb{I}(G, S, F, N, C)$ and a pacing strategy ρ.
output: A solution (if there exists one) to $\mathbb{I}$.

```
1  N* ← {n_1}
2  Create model M;
3  Relax all integrality constraints in M
4  Order S ;
5  p ← ρ ;
6  Let S* be the set of embedded NSs: set it to a set
7  while S* ≠ S and feasibility condition is respected do
8      Set IntSlices to the first ρ slices in S\S*
9      foreach s in IntSlices do
10         enforce integrality on all related variables in M
11     solve M
12     if a feasible solution is found then
13         foreach s in IntSlices do
14             Fix values on all related variables in M
15         Add IntSlices to S*
16         p ← ρ
17     else if S* ≠ ∅ then
18         Remove the last embedded slice from S*
19         p←p+1;
20     else
21         Stop: no feasible solution exists
22 if M is feasible then
23     PackNFSs() ;
24     return the complete solution to I
25 else
26     return no solution
```

fix values on their variables.

Steps 7-21 are the core of our proposed algorithm. First, let IntSlices be a set of slices whose variables will be enforced to be integers in the current round. This set is built regarding the p first slice requests in S and disregarding those already embedded in previous rounds (i.e., those whose variables have fixed values). Then, in step 11, the model $\mathbb{M}$ is solved. In this step, the variables associated with slice requests in S, S^*, and $IntSlice$ are respectively continuous, integer (or binary), and fixed (i.e., taking the values found in previous rounds where they were set to integer). Next, if $\mathbb{M}$ is feasible, steps 13-14 are responsible for fixing the values found in step 11 only on each slice request $s \in IntSlices$. Then, IntSlices is added to the embedded slices set S^* and p is restored to ρ. If $\mathbb{M}$ is not feasible and S^* is not empty (i.e., there is at least one slice embedded theretofore), then the last slice request is removed from the set S^* of embedded slices, and the value of p is incremented by one unit. This last step is responsible for creating a big set $IntSlices$ in step 8 during the next round.

Algorithm 7: PackNFSs()

input : An NSDP-DNF instance $\mathbb{I}(G, S, F, N, C)$ and a solution to ILP $\mathbb{M}$.
output: A feasible NFS-NF packing to $\mathbb{I}$.

```
1  Create a conflict graph G^c with no nodes
2  foreach active variable x in M do
3      add a related node to G^c
4  foreach each pair (u,v) of nodes in G^c do
5      add a related link to G^c if any packing constraint is violated ;  /* See Ineq. (8.3)
       */
6  maxClique ← 0
7  while maximum number of tries is not reached do
8      clique ← findClique(G^c) ;                                  /* by Gr-McD's algo.  */
9      if clique ≥ maxClique then
10         maxClique ← clique
11 minColoring ← ∞
12 while minimal chromatic number is not found or the maximum number of tries is not
     reached do
13     newColors ← Coloring(G^c);                                  /* by Syslo's algo.  */
14     if newColors ≤ minColoring then
15         minColoring ← newColors
16         if minColoring is equal to maxClique then
17             best solution found: stop and go to step 18
18 Pack NFSs into NF by the color of the related node in G^c
19 return NFS-NF packing
```

This loop stops when all slices are embedded or the instance $\mathbb{I}$ is proven to be unfeasible.

If a feasible solution is found, then *packNFSs()* procedure generates copies of network functions with different NFS types in order to process the data from all slices. This decision is made by translating the related NSDP-DNF sub-problem into a Vertex Coloring Problem [93] which is summarized in Algorithm 7. First, let $G^c = (V^c, E^c)$ be the conflict graph associated with the set of distributed NFSs as follows. A node v in V^c is associated with every tuple $(s, f, u) : s \in S, f \in F), u \in V$ and there exists an edge in $(v, v') \in E_c$ between any two nodes $v = (s, f, u)$ and $v' = (s', f', u')$ from V^c if $(q^s_{ff'} q^{s'}_{f'f} = 0) \vee (u \neq u')$ holds. Hence, an edge in G^c exists in order to forbid two NFSs to be packed together in the same network function n while violating anti-affinity packing constraints (see Inequalities (8.3)). Once the conflict graph is generated, its maximal clique size is calculated by applying the Grimmett-McDiarmid's greedy algorithm [128]: in each of its iteration, a random vertex is chosen and added to the current clique if and only if it is a common neighbor to all vertices already in the clique. In order to find a clique with maximal size, the greedy algorithm is run several times in a distributed parallel way (i.e., across multiple threads). The best value is then taken into consideration as lower-bound to the related vertex coloring problem, which is then solved by running the randomized

Table 8.1 – Time Complexity

Main Procedures	Steps	Asymptotic Complexity
Creating conflict graph	1-5	$O(SFV)$, V from physical network
Getting clique size	6-10	$O(V^4)$, V from conflict graph
Coloring conflict graph	11-17	$O(V^2)$, V from conflict graph

sequential coloring algorithm presented by Syslo [129]. It is also run several times in a distributed parallel way and returns the best coloring (i.e., with the minimal chromatic number). Regarding the related clique size previously calculated, the procedure stops any time an optimal coloring is found (i.e., the clique size is equal to the chromatic number) or a maximal number of tries is reached. Hence, each NFSs f represented by a vertex with the same color is packed into the same network function n.

8.3.1.1 Algorithm analysis

While Algorithm 6 is mainly driven by the solver's black box's time complexity, Table 8.1 depicts the complexity related to each inner procedure in Algorithm 7. In what follows, we present en discuss some important features of the proposed Relax-and-Fix algorithm.

Theorem 8.3.1. *If $IntSlice$ set is equal to the set S of slice requests and the mathematical formulation $\mathbb{M}$ is unfeasible, then no feasible solution exists for $\mathbb{I}$.*

Proof. If the pace p is equal to the number of slices requests in set S, then IntSlices will be equal to S in the next round. Hence, the integrality constraints will be enforced to all variables in $\mathbb{M}$. As it can only happen if there is no slice with fixed variables (see steps 17-19 in Algorithm 6) and applying the well-known Duality Theorem [133], it is clear that if $\mathbb{M}$ is not feasible, then no feasible solution exists to $\mathbb{I}$. □

As a direct implication of Theorem 8.3.1, we have the following corollary:

Corollary 1. *Relax-and-Fix always returns a feasible solution to $\mathbb{M}$ if there exists at least one.*

Proof. The proof is trivial, since that Relax-and-Fix will solve $\mathbb{M}$ enforcing the integrality constraints on all variables of the model in the worst case (i.e., $IntSlice$ set is equal to set S of slice requests). □

Theorem 8.3.2. *If S^* is empty and $\mathbb{I}$ is feasible, then the solution found to $\mathbb{M}$ is a lower-bound to ILP (8.1)-(8.17).*

Proof. The proof is trivial, since the Inequalities (3)-(4) and integrality constraints are partially relaxed and there is no variable with fixed value. □

8.3.2 Column generation-based algorithm

We now describe the Column Generation framework to solve the NSDP, which relies on the proposed E-ILP formulation (8.19)-(8.23) as the Master Problem. We first introduced the related dual formulation: the integrality constraints (8.23) are then relaxed and therefore respectively become $\pi_g^s \geq 0$. Then, we present an ILP formulation corresponding to the pricing sub-problem related to each slice request.

8.3.2.1 The restricted master problem

Since $G(s)$ has an exponential number of potentially suitable templates for each slice request, the E-ILP formulation (8.19)-(8.23) has an exponential number of variables. Hence, we propose to initialize the master problem with only a sub-set of columns (i.e., variables related to each $g \in G'(s) \subseteq G(s)$ for each $s \in S$). This model is then called *Restricted Master Problem* (RMP) and can be generated by proposing templates with no physical capacity requirement and cost with a large enough value (e.g., $\mu_g^s = 10^5$). We also propose to initialize the RMP with the final solution found by the proposed Relax-and-Fix algorithm described in the previous sections.

8.3.2.2 Pricing sub-problems

We first present the D-NCILP formulating related to the linear relaxation of E-ILP, whose dual variables are depicted on the right of each related constraint class in the previous section of this work.

$$max \sum_{s} \psi_s - \sum_{a \in A} b_a \phi_a - \sum_{u \in V} \sum_{c \in C} c_f^c \sigma_u^c \tag{8.24}$$

subject to

$$\psi_s - \sum_{a \in A} \bar{r}_a^g \phi_a - \sum_{f \in F} \sum_{n \in N} \sum_{u \in V} \sum_{c \in C} \bar{w}_{nu}^{gf} c_f^c \sigma_u^c \leq \mu_g^s \qquad , \forall s \in S, \forall g \in G(s) \tag{8.25}$$

$$\psi_s \quad \in \mathbb{R} \quad ,\forall s \in S \tag{8.26}$$

$$\phi_a \quad \in \mathbb{R}_+ \quad ,\forall a \in A \tag{8.27}$$

$$\sigma_u^c \quad \in \mathbb{R}_+ \quad ,\forall u \in V, \forall c \in C \tag{8.28}$$

The separation of inequalities (8.25) represents the pricing sub-problems related to the E-ILP formulation. Hence, the pricing sub-problem consists in finding a virtual network template for each slice request $s \in S$ in such a way that all technical constraints would be respected while improving the solution cost of RMP (i.e., decreasing the value of the objective function (8.19)). To this purpose, let $\bar{\psi}_s$ $\bar{\phi}_a$ $\bar{\sigma}_u^c$ be the components of the dual solution of the RMP related to constraints (8.20) and inequalities (8.21) and(8.22), respectively. Hence, the pricing problem consists of finding, for each slice request $s \in S$, a virtual network template g such that

$$\bar{\psi_s} - \sum_{a \in A} r_a^g \bar{\phi}_a - \sum_{f \in F} \sum_{n \in N} \sum_{u \in V} \sum_{c \in C} \bar{w}_{nu}^{gf} c_f^c \bar{\sigma}_u^c > \mu_g^s \tag{8.29}$$

The pricing sub-problem related to each slice request $s* \in S$ relies therefore on the compact formulation (8.1)-(8.17) whose objective function (8.11) is replaced by :

$$PSP(s^*) = \quad max \quad \bar{\psi}_{s*} - \sum_{n \in N} y_n^{s^*} - \sum_{a \in A} r_a \bar{\phi}_a - \sum_{f \in F} \sum_{n \in N} \sum_{u \in V} \sum_{c \in C} c_f^c w_{nu}^{s^* f} (\mu_u^c + \bar{\sigma}_u^c) \tag{8.30}$$

8.3.2.3 Column generation framework

Let us now describe the column generation-based framework proposed to solve the NSDP-DNF; which is summarized in Algorithm 8. First, let $G'(s) \subseteq G(s)$ be a minimal sub-set of potential virtual network templates for each slice request $s \in S$. Then, let E-ILP with these aforementioned sub-sets $G'(s)$ be our Restricted Master Problem, which is applied on its linear relaxation form. The algorithm starts by generation the sub-set $G'(s)$ for each slice request (see step 1 in Algorithm 8). In order to build such sub-sets, we apply the Relax-and-Fix as our primal heuristic. Then, in step 3, the related RMP are generated and solved. The next step is related to the pricing sub-problems, in which the PSP(s*) is solved for each slice request applying the values of the related dual variables from RMP. As previously discussed, this step is |S| times solved by applying the compact formulation (8.1)-(8.17), where $S = \{s*\}$ and whose objective function (8.11) is replaced by (8.30). If the reduced cost is

Algorithm 8: A Column Generation Framework to the NSDP-DNF

input : An NSDP-DNF instance $\mathbb{I}$ and a *G'(s)* for each slice request *s*.
output: A solution to $\mathbb{I}$.

```
1  Construct RMP with sub-sets G'(s)
2  while optimal solution is not found do
3      Solve relaxation of RMP ;
4      Get values from dual variables
5      foreach each slice request s* ∈ S do
6          Solve PSP(s*)
7          Get reduced cost ;
8          if reduced cost is negative then
9              Construct a virtual network template g
10             G'(s) ← G'(s) ∪ g
11     if no column is added to RMP then
12         Optimal solution found
13 Solve ILP of RMP ;
14 return the integer solution for I found in step 13.
```

negative, then the generated virtual network template is added to $G'(s)$ and the new RMP is solved with the new columns. These last steps are repeated until no column is generated from any PSP(s*). Finally, the ILP related to the RMP, with all generated columns and the integrality constraints (8.23), is then solved.

Note that this framework gets as input a minimal sub-set $G'(s)$ for each slice request. The construction of such sub-sets that ensure a feasible solution for the NSDP-DNF instance I is made in pre-processing.

8.4 Numerical experiments

Let us first detail the simulation settings. We propose different instance sizes (see Table 8.2), in which we set the processing capacity $cap(f)$ of each NFS in F^d to between 50% and 100% of the average volume generated by the traffic demands. For NFSs of F^c, this value was set to between 50% and 100% of the volume related to the total number n_s of expected UEs connected to the slice. Also, the total amount b_{fg} of traffic between two functions from $F(s) \cup G(s)$ was set to 1 Kbps per UE. As shown in Table 8.3, different instance classes are also proposed, which are related to the ratio between the resource required by the slices and available on the physical network, and also between the latency on the physical links and the threshold imposed by slices and pairs of NFSs. The complete reference of each generated instance is given by joining the acronyms of each size/class name from tables 8.2

Table 8.2 – Instance Sizes

Instance size	$\|V\|$	Graph Density*	$\|S\|$	$\|K\|$	$\|F^d\|$	$\|F^c\|$
Tiny (T)	10	0.15	2	1	2	2
Small (S)	15	0.10	2	2	2	2
Medium-Small (SM)	20	0.15	4	3	5	2
Medium (M)	25	0.15	4	8	5	5
Medium-Big (MB)	30	0.20	4	8	10	5
Big (B)	35	0.20	8	8	10	10

* Ratio between exiting and theoretically possible number of arcs.

Table 8.3 – Instance Classes

Latency	Description
Low (L)	The maximum latency d_{fg} between two NFSs and the end-to-end latency d_s imposed by each slice request $s \in S$ is set respectively to between 50% and 150% and to between 250% and 500% of the average latency on the physical links.
High (H)	The maximum latency d_{fg} between two NFSs and the end-to-end latency d_s of each slice $s \in S$ is set respectively to between 200% and 400% and to between 300% and 1000% of the average latency on the physical links.
Capacity	**Description**
Tight (T)	The available bandwidth b_a on the physical links have between 50% and 100% of the average volume (without compression) generated by the slices. In addition, each physical node $u \in V \backslash V^{ap}$ has enough capacity to host between 1 and 3 copies of each NFS type; application nodes has no available capacity.
Moderate (M)	The available bandwidth b_a on the physical links have between 200% and 300% of the average volume (without compression) generated by the slices. In addition, each physical node $u \in V \backslash V^{ap}$ has enough capacity to host between 5 and 8 copies of each NFS type; application nodes has no available capacity.
Isolation	**Description**
Weak (W)	10% of anti-affinity parameters q^s_{fg} are set to 0; they are randomly chosen.
Strong (S)	75% of anti-affinity parameters q^s_{fg} are set to 0; they are randomly chosen.

and 8.3. For example, $< S, L, M, S >$ refers to a *small* instance with *low* latency threshold, *moderate* capacity requirements, and *strong* isolation related to affinity constraints. We implemented our model in a Julia-JuMP environment using ILO CPLEX 12.10 as the linear solver. Our tests were run on a Linux server with an Intel Xeon E5-2650 CPU. Also, we provided 12 threads for each distributed parallel procedure. Finally, for each instance size/class, 30 different instances were randomly generated as previously discussed.

8.4.1 Compact formulation and relax-and-fix algorithm

We first analyze the efficiency of the proposed Relax-and-Fix algorithm. For this purpose, we compare it with the compact formulation (8.1)-(8.17) formulation, hereafter referred to as to *ILP*, introduced in this chapter. In these tests, we set the time limit to 3600 seconds (one hour). For

Table 8.4 – Relax-end-Fix: gap and runtime on different instances sizes.

	ILP		Relax-and-Fix	
Instance Size	**Gap (%)**	**Runtime (s)**	**Gap (%)**	**Runtime (s)**
Tiny	0.0	5±3	**1.14 ± 1.1**	**0.03±0.01**
Small	0.0	248±49	**1.65 ± 0.3**	**0.23±0.1**
Medium Small	26.5±2.3	3600*	8.44±0.2	**97±28**
Medium	48.4±28.8	3600*	**5.77±0.1**	**178±37**
Medium Big	82.8±32.2	3600*	**4.31±0.06**	**319±49**
Big	**	3600*	**5.90±0.1**	907±158

* Time limit reached. ** No feasible solution was found.

each instance size, 30 instances for each class (i.e., a combination of capacity, latency, and isolation constraints; see Table 8.3) were randomly generated as previously discussed.

Table 8.4 shows the execution time and final gap (average and standard deviation) on all instance sizes. While the second and third columns are related to the ILP formulation within the solver branch-and-bound, the two last columns depict the values when the proposed Relax-and-Fix algorithm was applied. In both cases, the final gap is related to the best lower bound between those obtained by Relax-and-Fix (see Theorem 8.3.2) and the solver on the ILP formulation (8.1)-(8.17). All results reported in Table 8.4 are related to random slice ordering and the 1-slice pacing strategy. First, we observe that the Relax-and-Fix was faster than ILP in all instance sizes. For instance, ILP reached the time limit on medium small-size instances and bigger ones, while our proposed Relax-and-Fix needed less than 20 minutes to solve any instance size. Indeed, we observed that approximately 90% of medium-small (resp. small) instances were solved in less than 1 (resp. 120) seconds. As seen in Table 8.4, the average gap could also be reduced from 82.8% to 4.31% (resp. 48.4% to 5.77%) on medium-big (resp. medium) instances. It is worth mentioning that 50% of medium instances had a gap smaller than 25%, and any feasible solution was found within 1-hour runtime for roughly 20% of medium-small instances. Moreover, while ILP could not find any feasible integer solution within 1-hour runtime for any big-size instance, the average runtime and final gap were respectively 907 seconds and 5.2% when Relax-and-Fix was applied on the same instances.

Fig. 8.1 shows the influence of different instance classes on the efficiency of our approach (random slice ordering and 1-slice pacing strategy were applied in all tests). Regarding all 32 possible combinations, we observe a moderate impact only on tiny and small instances (see Fig. 8.1a and Fig. 8.1b). Indeed, while 100% of $< T, H, M, W >$ (resp. $< S, H, M, W >$) instances had a gap smaller than 1.50% (resp. 2.50%), only 37.50% (resp. 60.0%) of $< T, L, T, S >$ (resp. $< S, L, T, S >$) instances

had the same solution quality. We also observed that the average final gap on tiny (resp. small) instances with tight capacity constraints was roughly 1.50% (resp. 2.47%) against 0.78% (resp. 0.50%) presented by those with moderate physical capacities. This behavior can be explained by the fact that almost all integrality constraints are relaxed during the first iterations of the proposed Relax-and-fix algorithm, which might potentially lead to the violation of several capacity constraints (see Inequalities (8.9)-(8.10)) before the last iteration (i.e., with all variables with either fixed values or related integrality constraints restored). However, we did not observe any important influence on the performance of our approach applying different instance classes on bigger instances (see Fig. 8.1c and Fig. 8.1d). Also, different latency and anti-affinity constraints did not statistically impact the quality of the final solutions on any instance size.

In what follows, we analyze the response of the Relax-and-Fix algorithm to different pacing and ordering strategies. To this purpose, we proposed three slice ordering strategies, which is responsible to chose the order of slices having the integrality constraints restored on the related variables (see steps 4 and 8-10 in Algorithm 6): by capacity (i.e., decreasing order of ratio between the capacity required by the slice and those available on the physical network), by latency (i.e., increasing order accepted latency on the data-plane function chain), and randomly. We also implemented different pacing strategies (from 1 to 4), which, in turn, is related to how many slices have their variables enforced to be integer on each iteration of our approach (see steps 5, 8, and 10 in Algorithm 6). For each combination ordering/pace, the average and the 95% confidence interval related to 100 instances are presented. These instances were randomly chosen among those generated with different sizes and classes (see Tables 8.2 and 8.3) used in the previous tests. Let us recall that the final gap is related to the best lower bound between those obtained by Relax-and-Fix and the solver on the ILP formulation.

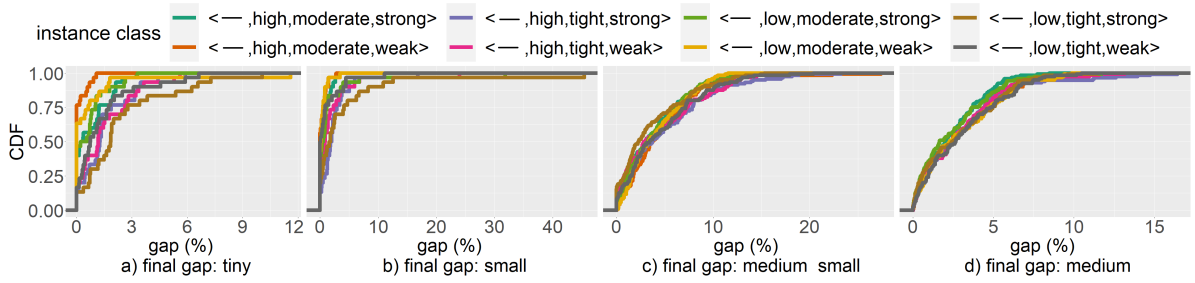


Figure 8.1 – Relax-end-Fix: qualitative analyses on different instance classes.

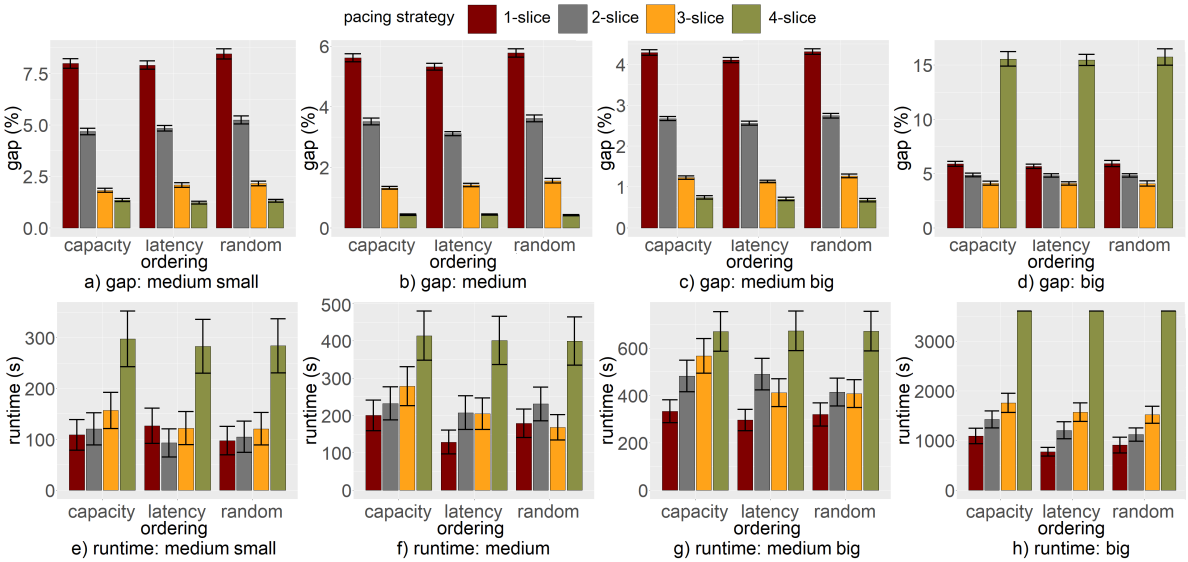


Figure 8.2 – Relax-and-Fix: ordering and pacing strategy trade-offs.

Fig. 8.2 presents the trade-off between runtime and quality of the final solution when different ordering and pacing strategies are applied within the proposed Relax-and-Fix framework. As expected, we observed a better solution quality when a bigger pacing is applied instances with 4 slices requests (see Fig. 8.2a, Fig. 8.2b, and Fig. 8.2c). For instance, the final gap could be reduced from 8.00% (resp. 6.92%) to 1.25% (resp. 0.40%) on medium-small (resp. medium) instances when latency (resp. random) ordering strategy was applied. Regarding all ordering strategies, the overall reduction on medium-big instances was approximately 22% from 1-slice pacing to 2-slice pace, 55% from 2-slice pacing to 3-slice pace, and 30% from 3-slice pacing to 4-slice pace.

In general, the runtime increases as the pacing strategy gets bigger. Indeed, the time needed to find the final solution by our approach increased from 270 seconds with 1-slice pacing to 490 seconds with 2-slice pacing on medium-big instances when capacity ordering strategy was applied (see Fig. 8.2g). We observe an even bigger increase from 3-slice pacing to 4-slice pacing in smaller instances (see Fig. 8.2e and Fig. 8.2f): in general, the runtime increased by 121% and 88% on medium small and medium instances, respectively. This behavior is expected since, even with fewer iterations (see the loop from steps 7-21 in Algorithm 6), the number of integer variables are considerably bigger on pacing strategies with more slices, which has a well-known strong influence on the branch-and-bound performed by the solver. This effect is even more noticeable on big-size instances (see Fig. 8.2d and Fig. 8.2h). While there is a smaller gain by increasing the pacing strategy, the runtime reached the 1-hour limit with the

4-slice pacing on all ordering strategies. As seen in Figures 8.2e and 8.2g, however, this is not always the case. For instance, we observed a decrease from 1-slice pacing to 2-slice pacing (resp. 2-slice pacing to 3-slice pacing) on medium-small (resp. medium) instances with latency (resp. random) ordering strategy. More tests must therefore be carried out to assess this behavior in such scenarios before conclusions can be drawn.

8.4.2 Extended formulation and column generation-based framework

We now present the results from solving the extended (8.19)-(8.23) along with the proposed column generation-based framework (see Algorithm 6). As a start point, we gave the solution found by our RF algorithm applying the 2-slice pace and capacity ordering strategies and set the time limit to 3600 seconds (1 hour). We refer to this approach as to *RF-CG*. Also, an artificial template-based starting point was proposed, in which an artificial column (i.e., slice template) was created with a large enough artificial cost (i.e., $\mu_g^s = 10^{2|S|}$) and no physical capacity required. As previously discussed, our pricing sub-problem is based on the compact formulation (8.1)-(8.17), hereafter referred to as to *CF*, where $S = \{s*\}$ and whose objective function (8.11) is replaced by (8.30). We refer this approach as to *AT-CG*. For each pricing iteration, we set the time limit to $3600 - \Delta$ seconds, where Δ is the time elapsed theretofore. Finally, for each instance size, 30 random instances were generated as previously discussed (see Tables 8.2 and 8.3). Table 8.5 depicts the result of our numerical experiments applying the proposed column generation framework on different instance sizes. For each approach, we present the linear relaxation gap[2], the final gap[3], the total runtime (in seconds), and the number of pricing iterations done before finding the best solution (only for the column generation-based approaches).

First, the final gap was substantially improved by the RF-CG on bigger instances. On medium-small and big instances, for example, the average reduction related to the CF was roughly 96% and 88%. This approach was also 96% (resp. 58%) faster than CF on tiny (resp. medium) instances and 44% faster than the AT-CG on average. In addition, while AT-CG could not find any feasible solution in less than 3600-second runtime on medium-big and big instances, RF-CG could improve the initial solution (i.e., those found by the relax-and-fix algorithm) in almost all instance sizes. Indeed, due

2. Calculated as the absolute value of the ratio between the found solution of the relaxed model and the best-known lower-bound, calculated as previously discussed.
3. Calculated as the absolute value of the ratio between the found solution of the related model with all integrality constraints and the best-known lower-bound, calculated as previously discussed.

Table 8.5 – Column Generation-based framework: gap and runtime on different instances sizes.

Approach		Instance Size					
		Tiny	Small	M. Small	Medium	M. Big	Big
Compact Formulation	**LR (%)**	22.5±6.9	22.6±1.2	47.2±7.0	42±5.1	38.5±16.2	39.1±13.4
	Gap (%)	0.0	0.0	26.5±2.3	48.4±28.8	82.8±32.2	**
	Runtime (s)	5±3	248±49	3600*	3600*	3600*	3600*
Relax-and-Fix + Column Generation	**LR (%)**	0.8±0.1	0.4±0.3	1.9±0.05	2.62±0.5	1.8 ±1.1	5.9±0.1
	Iterations	3.21±0.3	4.6±0.7	20.2±4.2	15.0±3.1	3.5±1.2	0
	Gap (%)	0.4±0.2	0.8+0.1	2.7±0.5	3.0±0.4	2.54±0.6	5.90 ±0.1
	Runtime (s)	0.2±0.01	4±2	492±200	1484±342	3600*	3600*
Artificial Temp. + Column Generation	**LR (%)**	0.8±0.1	0.4±0.3	1.9±0.5	0.2±0.07	15.6±9.2	27.3±16.4
	Iterations	5.46±0.5	6.7±0.9	21.9±3.7	18.5±2.8	2.2±1.3	0
	Gap (%)	3.5±0.5	10.2±5.6	16.5±6.1	47.2±10.3	**	**
	Runtime (s)	0.36±0.5	7±4	733±247	3600*	3600*	3600*

* Time limit reached. ** No feasible solution was found.

to time constraints, RF-CG could not complete any pricing iteration on big instances, generating no additional column (i.e., slice template) to the master problem. Compared to AT-CG, RF-CG could reach the best linear-relaxation solution with fewer iterations. For instance, RF-CG needed 41% (resp. 31%) fewer pricing iterations on tiny (resp. small) instances compared to AT-CG.

Moreover, we observe that the linear relaxation solutions of both CG-based frameworks were considerably better than those found by the compact formation: the average gap related to the best-known lower-bound could be reduced from roughly 22% (resp. 38%) to 0.4% (resp. 15%) on small (rep. medium-big) instances after less than 5 pricing iterations. It is important to mention that, the small number of pricing iterations on different instance sizes is due to different reasons. Indeed, while tiny and small instances could be solved in few iterations thanks to the quality of generated columns and the starting point (using the solution found by the relax-and-fix framework) on RF-CG, AT-CG instances could not be solved all potential pricing sub-problems to the optimality due to the time limit for bigger instances. For instance, the average gap of the last pricing iteration was approximately 13% (resp. 22%) on medium-big (resp. big) instances when the RF-CG was applied; these gaps were respectively 16% and 21% applying AT-CG. It is also worthwhile mentioning that we did not observe any significant improvement by changing the gap tolerance in the pricing routine since the solver spent a considerable amount of time on improving the dual bound in general on each pricing sub-problem.

8.5 Summary

In this chapter, we presented another variant of the NSDP, called Network Slice Design with Dedicated Network Functions, where the Hard Isolation policy is always applied, that is, all network

functions are slice-dedicated, keeping, however, the intra-slice anti-affinity constraints, which represent the possibility of packing two different NFSs into the same network function serving a given network slice. The motivation for keeping the NFS-NF mapping comes from decreasing the operational complexity related to the deployment of each NS. We then presented different strategies in order to efficiently solve the optimization problems related to the NSDP-DNF. First, we proposed a compact formulation and an extended one. To solve the former, we proposed a Relax-and-Fix heuristic that relies on repetitively solving the proposed related ILP with only a few integer variables and fixing or relaxing most of the remaining integer and binary ones. On the other hand, to address the exponential number of variables in the extended formulation, a column generation-based framework was then proposed.

In our simulations applying the proposed RF algorithm, for example, the average gap could be reduced from 82.8% to 4.31% (resp. 48.4% to 5.77%) on medium-big (resp. medium) instances for instance. Moreover, while ILP could not find any feasible integer solution within 1-hour runtime for any big-size instance, the average runtime and final gap were respectively 907 seconds and 5.2% when Relax-and-Fix was applied on the same instances. Regarding the column generation-based framework, this approach could improve the gap in all instances sizes, especially when the solution found by the RF algorithm was given as starting point. For instance, the gap and the runtime could be reduced by 96%.

Chapter 9

Concluding remarks and perspectives

We defined and studied the concept of device-to-device communications and network slicing in 5G systems and propose mathematical models and innovative algorithms to solve the underlying optimization problems. In particular, we studied the Domain Creation problem, which is a routing and resource assignment problem arising in future 5G networks. We proposed two algorithms, one exact and one heuristic, to solve it. The exact approach is based on a node-arc ILP formulation enhanced by two families of valid inequalities that are used within a branch-and-cut framework. We also presented a solving method based on a decomposition of the DCP into two sub-problems: the routing sub-problem and the resource allocation sub-problem. First, a significant impact of the cuts in strengthening the LP relaxation and reducing the computation time was observed. Despite a longer runtime for some instances using the proposed cuts, we could observe that the gap between the root solution and the final solution is always smaller when neighborhood cuts are applied. Using capacity cuts, this improvement is observed for half of the considered instances. However, the final size of the tree after finding the final solution using these cuts can be up to 76.40% smaller. Our experiments also showed that the proposed heuristic approach performs well, even on large instances with up to 2100 devices and 1500 service requests on 7-cell networks. It would be interesting and most probably very powerful to use it as a primal heuristic to boost the efficiency of an exact algorithm.

We also defined and modeled the network slice provisioning as an optimization problem including novel mapping and provisioning requirements. In particular, we considered novel mapping dimensions appearing with 5G systems, modeling the relationship between flexible radio access functional splitting, control-plane and data-plane function separation, and sharing policies. Different variants of the

problem were also considered and the proposed models are compliant with running standards. We demonstrated by numerical analyses the impact of taking into full and partial consideration of the peculiar constraints rising from the standards. For instance, we reported numerical results showing that flexible splitting appears as a key factor to deal with heterogeneous requirements to deploy distinct communication services, leading to considerable network slice cost decrease. In our simulations, the number of NFSs needed to deploy the virtual networks could be reduced by up to 56% depending on which of the six proposed sharing policies is applied to each network slice. We also observed that different variants related to the flexible splitting have an important impact on the physical network; depending on the selected approach, the average load on physical links could be reduced by a factor of 3.

In order to strengthen the linear relaxation of the proposed MILP, we presented several classes of valid inequalities and integrate them into a Branch-and-Cut framework to solve the problem. We further present several strategies to reduce the symmetries and the size of the model. Numerical experiments showed the efficiency of each approach on different instance classes. For instance, the proposed symmetry-breaking and lower bound-based constraints led to an important decrease in the final gap: from 30% to 1% in some instances. Also, our Branch-and-Cut algorithm could reduce the size of the Branch-and-Bound tree by 85% and outperformed the solver's Branch-and-Bound algorithm in all tests. Finally, our Row-Generation framework outperformed the Branch-and-Bound approach in all tests, especially in those with sparse graphs.

Moreover, to address the time complexity related to the proposed exact approaches, we presented an open-access framework based on a Math-heuristic to address the underlying optimization problem. The overall idea of the proposed approach relied on decomposing the NSDP into several sub-problems and sequentially solve them while encompassing control-plane and data-plane separation and novel mapping and decomposition dimensions influencing the placement and interconnection of slices. Numerical experiments showed the efficiency of our approach on different instance classes, which could attain near-optimal solutions in a competitive runtime. Comparing it to a mixed-integer linear programming formulation, the proposed Math-Heuristic could reduce the average runtime and the final gap by up to 78% and 90%, respectively. Moreover, our approach could reduce the congestion on the physical network, better balancing the data flow while considering all technical constraints. For instance, the average load on physical links and physical nodes could be reduced by 16% and 35%,

respectively.

We also studied the variant NSDP-DNF, where the Hard Isolation policy is always applied, that is, all network functions are slice-dedicated, ensuring however the intra-slice anti-affinity constraints, which represent the possibility of packing two different NFSs into the same network function serving a given network slice. We then proposed different strategies in order to efficiently solve the optimization problem. First, we presented a compact formulation and an extended one. To solve the former, we proposed a Relax-and-Fix heuristic that relies on repetitively solving the proposed related ILP with only a few integer variables and fixing or relaxing most of the remaining integer and binary ones. On the other hand, to address the exponential number of variables in the extended formulation, a column generation-based framework is then proposed. In our simulations applying the proposed Relax-and-Fix approach, for instance, the average gap could be reduced from 82.8% to 4.31% (resp. 48.4% to 5.77%) on medium-big (resp. medium) instances. Moreover, while ILP could not find any feasible integer solution within 1-hour runtime for any big-size instance, the average runtime and final gap were respectively 907 seconds and 5.2% when Relax-and-Fix was applied on the same instances. Regarding the column generation-based framework, this approach could improve the gap in all instances sizes, especially when the solution found by the RF algorithm was given as starting point. For instance, the gap and the runtime could be reduced by 96%.

On a practical note, a tough but interesting extension of our work related to the DCP is to include users' mobility and temporal aspect in radio resource assignment. Indeed, managing handovers is highly expected in real scenarios, which might potentially imply an important reconfiguration of the created domains in every short time period (i.e., minutes or even seconds). Also, other variants of the NSDP can be studied (e.g., applying Multi-Access Edge Computing) and a service-aware function objective might be proposed. In other words, different parameters might alternatively be optimized regarding the service related to each network slice (e.g., latency over capacity). Finally, an extension of our work on network slicing might also include availability constraints in order to ensure all technical constraints that are expected in the Service Level Agreements.

We also expect to embed the proposed column generation-based algorithm into a branch-and-price framework to ensure the optimal solution for each instance. On the other hand, in order to address the time complexity related to the column generation-based algorithm proposed to the NSDP-DNF, heuristic-based pricing algorithms might be proposed in order to attain near-optimal solutions in

a competitive runtime. Finally, applying clustering algorithms as pre-processing might also be a powerful approach to be applied in a distributed parallel programming, in which each thread might be responsible for solving smaller NSDP or DCP instances. For instance, such clustering algorithms can be based on geographical zone or even on service affinity.

Publications

In what follows, we present the published and submitted papers issued from this doctoral research.

Published papers

— A. Benhamiche, W.d.S. Coelho, and N. Perrot. "Routing and Resource Assignment Problems in Future 5G Radio Access Networks." International Network Optimization Conference (INOC). Avignon, France. 2019.

— W. d. S. Coelho, A. Benhamiche, N. Perrot and S. Secci, "Network Function Mapping: From 3G Entities to 5G Service-Based Functions Decomposition," in IEEE Communications Standards Magazine, vol. 4, no. 3, pp. 46-52, 2020.

— W. d. S. Coelho, A. Benhamiche, N. Perrot and S. Secci. "On the impact of novel function mappings, sharing policies, and split settings in network slice design." 2020 16th International Conference on Network and Service Management (CNSM). Izmir, Turkey. 2020.

— W. d. S. Coelho, A. Benhamiche, N. Perrot and S. Secci. "A Math-Heuristic for Network Slice Design", the 33nd International Teletraffic Congress (ITC), Avignon, France. 2021.

Accepted paper

— W. d. S. Coelho, A. Benhamiche, N. Perrot and S. Secci, "Function splitting, isolation, and placement trade-offs in network slicing". Transactions on Network and Service Management. November 2021, Preprint.

Submitted papers

— W. d. S. Coelho, A. Benhamiche, N. Perrot and S. Secci, "Exact Approaches for the Network Slice Design Problem". Submitted to Discrete Applied Mathematics Journal in September 2021.

— W. d. S. Coelho, A. Benhamiche, N. Perrot and S. Secci, "Relax-and-Fix and Column Generation Algorithms for the Network Slice Design Problem". Submitted to Networks Journal in September 2021.

Bibliography

[1] W. d. S. Coelho, A. Benhamiche, N. Perrot, and S. Secci, "Network function mapping: From 3G entities to 5G service-based functions decomposition," *IEEE Communications Standards Magazine*, vol. 4, no. 3, pp. 46–52, 2020.

[2] S. Abdelwahab, B. Hamdaoui, M. Guizani, and T. Znati, "Network function virtualization in 5G," *IEEE Comm. Mag.*, vol. 54, no. 4, pp. 84–91, 2016.

[3] T. Chen *et al.*, "Software defined mobile networks: concept, survey, and research directions," *IEEE Comm. Mag.*, vol. 53, no. 11, pp. 126–133.

[4] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5G: Survey and challenges," *IEEE Comm. Mag.*, vol. 55, no. 5, pp. 94–100, 2017.

[5] A. Benhamiche, W. da Silva Coelho, and N. Perrot, "Routing and resource assignment problems in future 5G radio access networks," *Network Optimization INOC 2019*, p. 89, 2019.

[6] 3rd Generation Partnership Project, "3GPP TS 23.214 V14.0.0: Architecture enhancements for control and user plane separation of EPC nodes (Release 14)," *Technical Specification Group Services and System Aspects*, 2016.

[7] ——, "3GPP TR 28.801 V15.1.0: Telecommunication management; Study on management and orchestration of network slicing for next generation (Release 15)," *Technical Specification Group Services and System Aspects*, 2018.

[8] ——, "3GPP TS 23.501 V15.4.0: System Architecture for the 5G System (Release 15)," *Technical Specification Group Services and System Aspects*, 2018.

[9] W. da Silva Coelho, A. Benhamiche, N. Perrot, and S. Secci, "On the impact of novel function mappings, sharing policies, and split settings in network slice design," in *International Conference on Network and Service Management*, 2020.

[10] ——, "A Math-Heuristic for network slice design," in *the 33nd International Teletraffic Congress*, Avignon, France, Aug. 2021.

[11] A. Gibbons, *Algorithmic graph theory*. Cambridge university press, 1985.

[12] K. Gutenschwager, S. Völker, A. Radtke, and G. Zeller, "The shortest path: Comparison of different approaches and implementations for the automatic routing of vehicles," in *Proceedings of the 2012 Winter Simulation Conference (WSC)*. IEEE, 2012, pp. 1–12.

[13] M. Dror, "Note on the complexity of the shortest path models for column generation in vrptw," *Operations Research*, vol. 42, no. 5, pp. 977–978, 1994.

[14] X. Zhang *et al.*, "Route selection for emergency logistics management: A bio-inspired algorithm," *Safety science*, vol. 54, pp. 87–91, 2013.

[15] X. Zhang, Y. Chen, and T. Li, "Optimization of logistics route based on dijkstra," in *2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS)*. IEEE, 2015, pp. 313–316.

[16] P. P. Pham and S. Perreau, "Performance analysis of reactive shortest path and multipath routing mechanism with load balance," in *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No. 03CH37428)*, vol. 1. IEEE, 2003, pp. 251–259.

[17] L. Zhang and S. Thomopoulos, "Neural network implementation of the shortest path algorithm for traffic routing in communication networks," in *International 1989 Joint Conference on Neural Networks*. IEEE, 1989, pp. 591–vol.

[18] C. Demetrescu, A. V. Goldberg, and D. S. Johnson, *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*. American Mathematical Soc., 2009, vol. 74.

[19] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[20] R. Bellman, "On a routing problem," *Quarterly of applied mathematics*, vol. 16, no. 1, pp. 87–90, 1958.

[21] D. B. Johnson, "Efficient algorithms for shortest paths in sparse networks," *Journal of the ACM (JACM)*, vol. 24, no. 1, pp. 1–13, 1977.

[22] S. Beamer, K. Asanovic, and D. Patterson, "Direction-optimizing breadth-first search," in *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, pp. 1–10.

[23] J. Y. Yen, "An algorithm for finding shortest routes from all source nodes to a given destination in general networks," *Quarterly of Applied Mathematics*, vol. 27, no. 4, pp. 526–530, 1970.

[24] J. Andrew, "Viterbi. 1967. error bounds for convolutional codes and an asymptotically optimal decoding algorithm," *IEEE Transactions on Information Theory*, vol. 13.

[25] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[26] M. F. Rego and H. G. Santos, "Algoritmos para o problema de coloração de grafos," *Computational optimization and applications*, vol. 19, no. 2, pp. 165–178, 2001.

[27] F. S. Hillier and G. J. Lieberman, *Introdução à pesquisa operacional*. McGraw Hill Brasil, 2013.

[28] L. V. Kantorovich, "The mathematical method of production planning and organization," *Management Science*, vol. 6, no. 4, pp. 363–422, 1939.

[29] G. B. Dantzig, "The diet problem," *Interfaces*, vol. 20, no. 4, pp. 43–47, 1990.

[30] J.-M. Boussard, J.-J. Daudin *et al.*, *La programmation linéaire dans les modèles de production*. Elsevier Mason SAS, 1988, no. 14.

[31] S. S. Rao, *Engineering optimization: theory and practice*. John Wiley & Sons, 2019.

[32] J. M. Borwein, "Characterization of optimality for the abstract convex program with finite dimensional range," *Journal of the Australian Mathematical Society*, vol. 30, no. 4, pp. 390–411, 1981.

[33] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[34] H. Moskowitz and G. P. Wright, *Investigación de operaciones*. Prentice hall, 1982, no. 658.57/M91oE.

[35] A. Schrijver, *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[36] R. Alves and C. Delgado, "Programação linear inteira," 1997.

[37] J. Clausen, "Branch and bound algorithms-principles and examples," *Department of Computer Science, University of Copenhagen*, pp. 1–30, 1999.

[38] M. J. T. G. Alves, "Apoio à decisão em problemas de programação inteira e inteira-mista multiobjectivo: contribuições metodológicas," Ph.D. dissertation, 2001.

[39] M. Grötschel, L. Lovász, and A. Schrijver, "The ellipsoid method and its consequences in combinatorial optimization," *Combinatorica*, vol. 1, no. 2, pp. 169–197, 1981.

[40] C. Barnhart *et al.*, "Branch-and-price: Column generation for solving huge integer programs," *Operations research*, vol. 46, no. 3, pp. 316–329, 1998.

[41] P. C. Gilmore and R. E. Gomory, "A linear programming approach to the cutting stock problem—part ii," *Operations research*, vol. 11, no. 6, pp. 863–888, 1963.

[42] L. Sarah, L. Proll, and A. Wren, "A column generation approach to bus driver scheduling," in *Proceedings of the 4th Meeting of the EURO Working Group on Transportation*, 1996, pp. 36–52.

[43] G. C. Pileggi, R. Morabito, and M. N. Arenales, "Heurísticas para os problemas de geração e sequenciamento de padrões de corte bidimensionais," *Pesquisa operacional*, vol. 27, no. 3, pp. 549–568, 2007.

[44] J. E. C. Arroyo *et al.*, "Heurísticas e metaheurísticas para otimização combinatória multiobjetivo," 2002.

[45] W. J. Clancey, "Heuristic classification," *Artificial intelligence*, vol. 27, no. 3, pp. 289–350, 1985.

[46] H. A. Simon and A. Newell, "Heuristic problem solving: The next advance in operations research," *Operations research*, vol. 6, no. 1, pp. 1–10, 1958.

[47] I. R. Katz *et al.*, *Proceedings of the SIGCHI conference on human factors in computing systems*. ACM Press/Addison-Wesley Publishing Co., 1995.

[48] 3rd Generation Partnership Project, "Release 1999: Specifications," *Technical Specification Group Services and System Aspects*, 1999.

[49] M. Neruda and R. Bestak, "Evolution of 3gpp core network," in *2008 15th International Conference on Systems, Signals and Image Processing*. IEEE, 2008, pp. 25–28.

[50] 3rd Generation Partnership Project, "3GPP TS 21.101 V8.0.0 : Technical Specifications and Technical Reports for a UTRAN-based 3GPP system (Release 8)," *Technical Specification Group Services and System Aspects*, 2009.

[51] ——, "3GPP TS 23.203 V11.9.0: Policy and charging control architecture (Release 11)," *Technical Specification Group Services and System Aspects*, 2013.

[52] ——, "3GPP TS 32.291 V15.2.1 : Telecommunication management, Charging management, 5G system and charging service (Release 15)," *Technical Specification Group Services and System Aspects*, 2019.

[53] ——, "3GPP TS 29.521 V15.3.0: 5G System; Binding Support Management Service (Release 15)," *Technical Specification Group Services and System Aspects*, 2019.

[54] ——, "3GPP TS 23.502 V15.4.0 : Procedures for the 5G System (Release 15)," *Technical Specification Group Services and System Aspects*, 2018.

[55] ——, "3GPP TS 29.503 V16.0.0 : 5G System; Unified Data Management Services (Release 16)," *Technical Specification Group Services and System Aspects*, 2019.

[56] O. Chabbouh, S. B. Rejeb, N. Agoulmine, and Z. Choukair, "Cloud RAN architecture model based upon flexible RAN functionalities split for 5G networks," in *WAINA 2017*.

[57] 3rd Generation Partnership Project, "3GPP TR 38.801 V14.0.0 :Study on new radio access technology ," 2017.

[58] C. Mobile, "C-RAN: the road towards green RAN," *White paper, ver*, vol. 2, pp. 1–10, 2011.

[59] L. M. Larsen, A. Checko, and H. L. Christiansen, "A survey of the functional splits proposed for 5G mobile crosshaul networks," *IEEE Comm. Surveys & Tutorials*, vol. 21, no. 1, pp. 146–172, 2018.

[60] K. Katsalis, N. Nikaein, and A. Huang, "Jox: An event-driven orchestrator for 5g network slicing," in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018, pp. 1–9.

[61] Mobile and wireless communications Enablers for the Twenty-twenty Information Society, "3GPP TR 38.913 V14.3.0 : "Final Report on Architecture," *ICT-317669-METIS/D6.4*, 2015.

[62] P. Marsch *et al.*, "5g radio access network architecture: Design guidelines and key considerations," *IEEE Comm. Mag.*, vol. 54, no. 11, 2016.

[63] A. Maeder *et al.*, "Towards a flexible functional split for cloud-ran networks," in *2014 EuCNC*. IEEE, 2014, pp. 1–5.

[64] X. Wang *et al.*, "Centralize or distribute? a techno-economic study to design a low-cost cloud radio access network," in *ICC*. IEEE, 2017.

[65] A. Asadi, Q. Wang, and V. Mancuso, "A survey on device-to-device communication in cellular networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 1801–1819, 2014.

[66] R. H. Tehrani *et al.*, "Licensed spectrum sharing schemes for mobile operators: A survey and outlook," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2591–2623, 2016.

[67] J. Jeon *et al.*, "Lte in the unlicensed spectrum: Evaluating coexistence mechanisms," in *2014 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2014, pp. 740–745.

[68] Y.-D. Lin and Y.-C. Hsu, "Multihop cellular: A new architecture for wireless communications," in *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)*, vol. 3. IEEE, 2000, pp. 1273–1282.

[69] G. Fodor *et al.*, "Design aspects of network assisted device-to-device communications," *IEEE Communications Magazine*, vol. 50, no. 3, pp. 170–177, 2012.

[70] B. Zhou, H. Hu, S.-Q. Huang, and H.-H. Chen, "Intracluster device-to-device relay algorithm with optimal resource utilization," *IEEE transactions on vehicular technology*, vol. 62, no. 5, pp. 2315–2326, 2013.

[71] J. C. Li, M. Lei, and F. Gao, "Device-to-device (d2d) communication in mu-mimo cellular networks," in *2012 IEEE global communications conference (GLOBECOM)*. IEEE, 2012, pp. 3583–3587.

[72] N. Golrezaei, P. Mansourifard, A. F. Molisch, and A. G. Dimakis, "Base-station assisted device-to-device communications for high-throughput wireless video networks," *IEEE Transactions on Wireless Communications*, vol. 13, no. 7, pp. 3665–3676, 2014.

[73] N. Golrezaei, A. G. Dimakis, and A. F. Molisch, "Device-to-device collaboration through distributed storage," in *2012 IEEE global communications conference (GLOBECOM)*. IEEE, 2012, pp. 2397–2402.

[74] A. Asadi and V. Mancuso, "Wifi direct and lte d2d in action," in *2013 IFIP Wireless Days (WD)*. IEEE, 2013, pp. 1–8.

[75] M. Ji, G. Caire, and A. F. Molisch, "Wireless device-to-device caching networks: Basic principles and system performance," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 1, pp. 176–189, 2015.

[76] H. Cai, I. Koprulu, and N. B. Shroff, "Exploiting double opportunities for deadline based content propagation in wireless networks," in *2013 Proceedings IEEE INFOCOM*. IEEE, 2013, pp. 764–772.

[77] Q. Wang, B. Rengarajan, and J. Widmer, "Increasing opportunistic gain in small cells through base station-driven traffic spreading," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*. IEEE, 2014, pp. 1–9.

[78] Q. Wang and B. Rengarajan, "Recouping opportunistic gain in dense base station layouts through energy-aware user cooperation," in *2013 IEEE 14th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE, 2013, pp. 1–9.

[79] C.-H. Yu, K. Doppler, C. B. Ribeiro, and O. Tirkkonen, "Resource sharing optimization for device-to-device communication underlaying cellular networks," *IEEE Transactions on Wireless communications*, vol. 10, no. 8, pp. 2752–2763, 2011.

[80] J. Wang *et al.*, "Resource sharing of underlaying device-to-device and uplink cellular communications," *IEEE Communications Letters*, vol. 17, no. 6, pp. 1148–1151, 2013.

[81] H. S. Chae, J. Gu, B.-G. Choi, and M. Y. Chung, "Radio resource allocation scheme for device-to-device communication in cellular networks using fractional frequency reuse," pp. 58–62, 2011.

[82] H. Wang and X. Chu, "Distance-constrained resource-sharing criteria for device-to-device communications underlaying cellular networks," *Electronics letters*, vol. 48, no. 9, pp. 528–530, 2012.

[83] S. R. Chowdhury, M. A. Salahuddin, N. Limam, and R. Boutaba, "Re-architecting nfv ecosystem with microservices: State of the art and research challenges," *IEEE Network*, vol. 33, no. 3, pp. 168–176, 2019.

[84] I. Alawe *et al.*, "On the scalability of 5g core network: the amf case," in *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2018, pp. 1–6.

[85] K. I. Aardal, A. Hipolito, S. P. M. van Hoesel, and B. Jansen, "A branch-and-cut algorithm for the frequency assignment problem," *Research Memorandum 96/011, Maastricht University*, 1996.

[86] K. I. Aardal *et al.*, "Models and solution techniques for frequency assignment problems," *Annals of Operations Research*, vol. 153, no. 1, pp. 79–129, Sep 2007.

[87] B. Jaumard, C. Meyer, and B. Thiongane, *ILP Formulations for the Routing and Wavelength Assignment Problem: Symmetric Systems*. Springer US, 2006, pp. 637–677.

[88] A. E. Ozdaglar and D. P. Bertsekas, "Routing and wavelength assignment in optical networks," *IEEE/ACM Trans. Netw.*, vol. 11, no. 2, pp. 259–272, April 2003.

[89] M. Zulhasnine, C. Huang, and A. Srinivasan, "Efficient resource allocation for device-to-device communication underlaying lte network," *2010 IEEE 6th International conference on wireless and mobile computing, networking and communications*, pp. 368–375, 2010.

[90] A. Capone, L. Chen, S. Gualandi, and D. Yuan, "A new computational approach for maximum link activation in wireless networks under the sinr model," *IEEE transactions on wireless communications*, vol. 10, no. 5, pp. 1368–1372, 2011.

[91] F. Margot, *Symmetry in Integer Linear Programming*, 2010, pp. 647–686.

[92] G. L. Nemhauser and G. Sigismondi, "A Strong Cutting Plane/Branch-and-Bound Algorithm for Node Packing," *Journal of the Operational Research Society*, vol. 43, no. 5, pp. 443–457, 1992.

[93] E. Malaguti and P. Toth, "A survey on vertex coloring problems," *International transactions in operational research*, vol. 17, no. 1, pp. 1–34, 2010.

[94] D. W. Matula, G. Marble, and J. D. Isaacson, "Graph coloring algorithms," pp. 109–122, 1972.

[95] M. Depolli *et al.*, "Exact parallel maximum clique algorithm for general and protein graphs," *Journal of chemical information and modeling*, vol. 53, no. 9, pp. 2217–2228, 2013.

[96] "3rd Generation Partnership Project. Release 16: 5G system - phase 2," https://www.3gpp.org/release-16, Accessed: 2020-07-24.

[97] A. Baumgartner, T. Bauschert, A. M. Koster, and V. S. Reddy, "Optimisation models for robust and survivable network slice design: A comparative analysis," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 2017, pp. 1–7.

[98] B. Tan *et al.*, "Analog coded softcast: A network slice design for multimedia broadcast/multicast," *IEEE Transactions on Multimedia*, vol. 19, no. 10, pp. 2293–2306, 2017.

[99] F. Malandrino and C.-F. Chiasserini, "Getting the most out of your vnfs: Flexible assignment of service priorities in 5G," in *WoWMoM 2019*.

[100] T. Truong-Huu, P. M. Mohan, and M. Gurusamy, "Service chain embedding for diversified 5G slices with virtual network function sharing," *IEEE Comm. Letters*, vol. 23, no. 5, pp. 826–829, 2019.

[101] M. R. Crippa *et al.*, "Resource sharing for a 5G multi-tenant and multi-service architecture," in *European Wireless Conference*. VDE, 2017.

[102] I. Alawe *et al.*, "Smart scaling of the 5G core network: an rnn-based approach," in *2018 GLOBECOM*. IEEE, 2018, pp. 1–6.

[103] X. Wang, C. Wu, F. Le, and F. C. Lau, "Online learning-assisted vnf service chain scaling with network uncertainties," in *2017 CLOUD*. IEEE, 2017, pp. 205–213.

[104] X. Fei, F. Liu, H. Xu, and H. Jin, "Adaptive VNF scaling and flow routing with proactive demand prediction," in *INFOCOM*. IEEE, 2018, pp. 486–494.

[105] X. Zhang, C. Wu, Z. Li, and F. C. Lau, "Proactive VNF provisioning with multi-timescale cloud resources: Fusing online learning and online optimization," in *IEEE INFOCOM 2017*. IEEE, 2017, pp. 1–9.

[106] T. C. Hu, "Multi-commodity network flows," *Operations research*, vol. 11, no. 3, pp. 344–360, 1963.

[107] G. Wang, G. Feng, S. Qin, and R. Wen, "Efficient traffic engineering for 5G core and backhaul networks," *J. of Comm. and Networks*, vol. 19, no. 1, pp. 80–92, 2017.

[108] N. Molner, A. De la Oliva, I. Stavrakakis, and A. Azcorra, "Optimization of an integrated fronthaul/backhaul network under path and delay constraints," *Ad Hoc Networks*, vol. 83, 2019.

[109] A. Fischer *et al.*, "Virtual network embedding: A survey," *IEEE Comm. Surveys & Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[110] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*. IEEE, 2015, pp. 171–177.

[111] J. J. A. Esteves, A. Boubendir, F. Guillemin, and P. Sens, "Location-based data model for optimized network slice placement," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2020, pp. 404–412.

[112] ——, "Heuristic for edge-enabled network slicing optimization using the "power of two choices"," in *2020 16th International Conference on Network and Service Management (CNSM)*. IEEE, 2020, pp. 1–9.

[113] A. Fendt, C. Mannweiler, L. C. Schmelz, and B. Bauer, "A formal optimization model for 5g mobile network slice resource allocation," in *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE, 2018, pp. 101–106.

[114] J. Liu *et al.*, "Provisioning optimization for determining and embedding 5g end-to-end information centric network slice," *IEEE Transactions on Network and Service Management*, 2020.

[115] M. Rost and S. Schmid, "On the hardness and inapproximability of virtual network embeddings," *IEEE/ACM Transactions on Networking*, 2019.

[116] D. Bhamare, R. Jain, M. Samaka, and A. Erbad, "A survey on service function chaining," *J. of Network and Computer Applications*, vol. 75, pp. 138–155, 2016.

[117] N. G. M. N. Alliance, "5G White Paper," 2015.

[118] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0–Survivable Network Design Library," in *Proceedings of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium*, April 2007, http://sndlib.zib.de.

[119] W. da Silva Coelho. (2020) NSDP: source code and instances. [Online]. Available: https://github.com/wdscoelho/NSDP

[120] L. Ford and D. Fulkerson, "«maximal flow through a network», canadian journal of mathematics," 1956.

[121] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Communications of the ACM*, vol. 16, no. 9, pp. 575–577, 1973.

[122] JuMP, *Julia for Mathematical Programming: Callbacks*, 2020, [Online]. Available on: https://jump.dev/JuMP.jl/v0.21.1/callbacks. Accessed: August 24, 2020.

[123] W. da Silva Coelho. (2021) OptNSDP: source code and instances. [Online]. Available: https://github.com/wdscoelho/OptNSDP

[124] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *Journal of the ACM (JACM)*, vol. 19, no. 2, pp. 248–264, 1972.

[125] E. A. Dinic, "Algorithm for solution of a problem of maximum flow in networks with power estimation," in *Soviet Math. Doklady*, vol. 11, 1970, pp. 1277–1280.

[126] Y. Boykov and V. Kolmogorov, "An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 9, pp. 1124–1137, 2004.

[127] Solving a MIP with distributed parallel optimization. IBM. [consulted on 18th December 2020]. Available on: https://www.ibm.com/docs/en/icos/20.1.0?topic=optimization-solving-mip-distributed-parallel.

[128] G. R. Grimmett and C. J. McDiarmid, "On colouring random graphs," in *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 77, no. 2. Cambridge University Press, 1975, pp. 313–324.

[129] M. M. Sysło, "Sequential coloring versus welsh-powell bound," *Discrete mathematics*, vol. 74, no. 1-2, pp. 241–243, 1989.

[130] W. da Silva Coelho. (2021) A Math-Heuristic for the NSDP: source code and instances. [Online]. Available: https://github.com/wdscoelho/HeuristicNSDP

[131] F. Malandrino, C. F. Chiasserini, G. Einziger, and G. Scalosub, "Reducing service deployment cost through vnf sharing," *IEEE/ACM Transactions on Networking*, vol. 27, no. 6, pp. 2363–2376, 2019.

[132] N. G. M. N. Alliance, "Description of Network Slicing Concept," 2016.

[133] A. Bachem and W. Kern, "Linear programming duality," in *Linear Programming Duality*. Springer, 1992, pp. 89–111.

Résumé de Thèse

Les flux de données sur les réseaux mobiles a une croissance très accélérée avec des demandes de plus en plus importantes depuis les années 2000. En quelques années, ces réseaux pourraient ainsi atteindre leurs capacités maximales en termes de transmission de données. Pour faire face à tous ces défis, la technologie 5G se pose en permettant la numérisation de la société et de l'information économique. L'idée derrière le concept de la 5G est qu'elle ne correspond pas à une simple augmentation du débit, comme c'était le cas pour les générations précédentes, mais aussi qu'il s'agit par là d'élargir la diversité des équipements des utilisateurs.

Cette évolution technologique touchera l'ensemble de l'environnement réseau, allant des accès cellulaires et radio aux architectures de services applicatifs. Cette transition remet en question la conception du réseau, car plusieurs ressources et segments, historiquement gérés de manière indépendante, doivent être exploités à la fois avec une continuité dans l'allocation des ressources réseau et informatique et la proposition d'un service global et unique. Dans ce contexte, différents fournisseurs peuvent être associés à différents services de communication exécutés sur le même réseau physique au niveau des segments d'accès, de cœur et d'application. Pour fournir le provisionnement flexible nécessaire, les technologies Network Function Virtualization [1], Software Defined Networking [2] et Network Slicing [3] peuvent être adoptées pour permettre au fournisseur CS de déployer ses services sur le dessus. de réseaux logiques.

Pour faire face à ces défis, 'Network Slice' couvre non seulement l'abstraction au niveau de l'application mais également la virtualisation des couches physiques et de commutation, avec différentes technologies d'accès radio et de communication de liaison. Ainsi, chaque fournisseur de services doit pouvoir déployer ses services de communication avec des réseaux logiques, appelés Network Slices, spécifiquement adaptés à ses exigences techniques. Le mode de communication

Device-to-Device (D2D) permet aussi de réutiliser les ressources radio et de diminuer la latence de bout en bout des communications locales. D2D permettrait à un ensemble d'utilisateurs géographiquement proches les uns des autres d'établir des communications D2D directes pour accéder à un service donné.

Dans ce contexte, l'optimisation des ressources dans les réseaux cellulaires devient cruciale sur le dimensionnement des réseaux de backhauling, et donc sur le placement des fonctions de cœur de réseau et la configuration des serveurs d'applications edge computing. De plus, différentes politiques de partage et de mise à l'échelle des fonctions de contrôle par rapport au plan de données devraient être appliquées.

L'objectif global de ce travail est donc de définir et d'étudier le concept de conception d'D2D et de Network Slicing dans les systèmes 5G et de proposer des modèles mathématiques et des algorithmes innovants pour résoudre les problèmes d'optimisation sous-jacents.

1.1 La communication appareil à appareil

Un *domaine* est défini comme l'ensemble des utilisateurs et des stations de base qui sont utilisés pour établir des communications mobiles (D2D ou cellulaires) liées à un service spécifique. Deux utilisateurs peuvent alors se communiquer via des liaisons cellulaires, en utilisant les liaisons classiques ou D2D, et les deux technologies peuvent coexister au sein du même réseau mobile. Dans tous les cas, des ressources radio devraient être allouées à chaque liaison active impliquée dans une communication, et le niveau SINR (Signal-to-Interference-plus-Noise Ratio) requis par le service devrait être assuré.

1.1.1 Définition du problème

Nous considérons un réseau mobile composé d'un ensemble de dispositifs, d'un ensemble d'antennes et d'un ensemble de services éligibles aux communications D2D, avec leurs matrices de trafic associées. Un poids non négatif, correspondant au SINR, est associé à chaque lien. C'est une mesure de la qualité de la communication qui pourrait être établie à l'aide de ce lien. Chaque service nécessite un seuil de qualité minimal en termes de SINR et de ressources disponibles (capacité matérielle pour les appareils, ressources radio pour les liens) pour être

établi avec succès. Une fois ces conditions remplies, les services sont délivrés via les différents types de ressources allouées à la fois aux appareils et aux liens.

Dans ce contexte, nous définissons le problème de création de domaine (DCP) comme suit.

Definition 1.1.1. *Le problème de création de domaine consiste à trouver une allocation de coût minimum des ressources radio aux arcs actifs pour fournir un chemin de routage réalisable pour chaque demande. En particulier, un routage* $p^k = \{(o^k, u), \ldots, (v, d^k)\}$ *pour une demande* k *est dit faisable si*

- *tous les arcs de* p^k *ont une valeur SINR supérieure au seuil de qualité requis par la demande* k *et,*
- *tous les nœuds de* p^k *ont une capacité suffisante pour satisfaire les besoins en ressources.*

1.1.2 Sommaire

Nous avons étudié le problème de création de domaine en proposant deux algorithmes: exact et heuristique. L'approche exacte est basée sur une formulation ILP nœud-arc renforcée par deux familles d'inégalités valides qui sont utilisées dans un cadre de branch-and-cut. Les résultats montrent un impact significatif des coupures dans le renforcement de la relaxation LP et la réduction du temps de calcul. Nous nous attendons à ce que l'ajout de classes de coupes supplémentaires et la réalisation d'une analyse pour découvrir les spécificités des instances difficiles (quelle que soit leur taille) permettront de résoudre des instances encore plus grandes. Une question naturelle serait d'envisager une formulation non compacte, basée sur des variables de chemin et de proposer un algorithme basé sur la génération de colonnes pour la résoudre. D'un autre côté, nos expériences montrent que l'approche heuristique fonctionne bien, même sur de grandes instances avec jusqu'à 2100 appareils et 1500 demandes de service sur des réseaux à 7 cellules. Il serait intéressant et très probablement très puissant de l'utiliser comme heuristique primale pour augmenter l'efficacité d'un algorithme exact.

1.2 Le problème Network Slice Design

Le système 5G couvre désormais non seulement l'abstraction de *slice* au niveau de l'application comme cela a été fait avec les travaux préliminaires sur le 'slicing', mais aussi la virtualisation des

couches physiques et de commutation, avec différentes technologies d'accès radio et de communication de liaison. Dans ce contexte, différents fournisseurs peuvent être associés à différents services de communication s'exécutant sur le même réseau physique au niveau des segments d'accès, de cœur et d'application.

1.2.1 Définition du problème

Nous définissons le problème de Network Slice Design comme suit.

Definition 1.2.1. *Étant donné un graphe orienté G représentant le réseau physique, un ensemble de requêtes de slices S, un ensemble de requêtes de trafic $K(s)$ associé à chaque requête $s \in S$, et un ensemble F de types de NFS, le NSDP consiste à déterminer le nombre de NFS à installer pour chaque $s \in S$, la taille des NF qui les hébergent ainsi qu'à décider s'ils doivent être installés de manière centralisée ou distribuée (c'est-à-dire en sélectionnant le découpage fonctionnel), de sorte que (i) les demandes de $K(s)$ puissent être contrôlées et acheminées dans G en utilisant ces NF ; (ii) les NFS installés sur G peuvent être compressés dans les NF tout en satisfaisant à la fois les contraintes d'isolement et de capacité ; et (iii) un chemin en G est associé à chaque paire de NF installées qui doivent être connectées. L'objectif est de concevoir chaque slice et de les intégrer dans le réseau physique G tout en minimisant le coût total de déploiement des requêtes de slice et en garantissant toutes les contraintes techniques imposées par les couches physiques et virtuelles.*

1.2.2 Sommaire

Nous avons démontré par simulation l'impact de la prise en compte totale et partielle des contraintes particulières découlant des normes. Nous avons rapporté des résultats numériques montrant que le fractionnement flexible apparaît comme un facteur clé pour faire face à des exigences hétérogènes pour déployer des services de communication distincts, conduisant à une diminution considérable des coûts de *slices*. Dans nos simulations, le nombre de NFS nécessaires pour déployer les réseaux virtuels pourrait être réduit jusqu'à 56% selon laquelle des six politiques de partage proposées est appliquée à chaque *slice*. Nous avons également observé que différentes variantes liées au fractionnement flexible ont un impact important sur le réseau physique: selon l'approche choisie, la charge sur les liens physiques pourrait être réduite d'un facteur 3.

1.3 Approches exactes pour le NSDP

Nous proposons plusieurs approches exactes basées sur la formulation MILP pour le problème, qui comprend les nouvelles contraintes *splitting*, *mapping* et *provisioning* décrites dans les documents de normes 5G publiés [4, 5, 6]. Nous proposons plusieurs classes d'inégalités valides afin de renforcer la relaxation linéaire du MILP proposé et de les intégrer dans un cadre Branch-and-Cut pour résoudre le problème. Nous présentons en outre plusieurs stratégies pour réduire les symétries et la taille du modèle.

1.3.1 Contraintes de rupture de symétrie

Les inégalités suivantes sont des contraintes de rupture de symétrie pour le NSDP.

$$x_{nu}^{sf} \leq \sum_{t \in S} \sum_{g \in F} \sum_{v \in V} x_{n-1v}^{tg} \qquad \forall s \in S, \forall f \in F, \forall u \in V, \forall n \in N \backslash \{n_1\} \tag{1.1}$$

1.3.2 Inégalités valides

Nous présentons aussi plusieurs familles d'inéquations valides utilisées pour renforcer la relaxation linéaire de la formulation proposée.

1.3.2.1 Inégalité de la borne inférieure

La première inégalité exprime une borne inférieure sur le nombre de NFS nécessaires pour satisfaire toutes les demandes de slices et les demandes de trafic associées.

The following inequality

$$\sum_{f \in F^c} \left\lceil \sum_{s \in S} \frac{n_s b_f}{cap(f)} \right\rceil + \sum_{f \in F^d} \left\lceil \sum_{k \in K(s): s \in S} \frac{\lambda_{f-1} b^k}{cap(f)} \right\rceil \leq \sum_{f \in F} \sum_{n \in N} \sum_{v \in V} y_{nu}^f \tag{1.2}$$

is valid for the NSDP.

1.3.2.2 Inégalités basées sur le plus court chemin

Les inégalités valides triviales suivantes sont basées sur le fait que le chemin physique assigné pour transporter le flux associé à une demande de trafic ne peut pas être plus court que le plus

court chemin entre son origine et sa destination.

$$\sum_{a \in A_p} d_a(\gamma^{ka}_{f_{|F^d|}f_0} + \sum_{f \in \{f_0\} \cup F^d \setminus \{f_{|F^d|}\}} \gamma^{ka}_{ff+1}) \geq sp(k) \qquad ,\forall k \in K(s) : s \in S \tag{1.3}$$

are valid for the NSDP problem.

1.3.2.3 Inégalités basées sur les coupes minimales

Les inégalités valides suivantes sont basées sur un théorème *min-cut/max-flow* de Ford et Fulkerson [7], qui précise que pour un même produit, le débit maximal est égal à la coupe minimale séparant les nœuds d'origine et destination associés. Nous ne considérons donc que les arcs ayant une capacité suffisante pour transporter individuellement le flux attendu.

$$\sum_{a \in \delta}(\gamma^{ka}_{f_{|F^d|}f_0} + \sum_{f \in \{f_0\} \cup F^d \setminus \{f_{|F^d|}\}} \gamma^{ka}_{ff+1}) \geq 1 \qquad \forall k \in K(s) : s \in S, \forall \delta \in \Delta(k) \tag{1.4}$$

are valid for the NSDP problem.

1.3.3 Un algorithme de génération de lignes pour le NSDP

Nous proposons également l'algorithme *Reduced MILP-based Row-Generation* (RMRG), qui est basé sur un modèle relaxé : la formulation originale proposée est réduite à une nouvelle sans les contraintes d'isolement, de capacité et de latence. Ainsi, cette nouvelle formulation n'a que les contraintes liées aux principales décisions sur le NSDP : les inégalités de sélection fractionnée, les équations de dimensionnement, les inégalités d'emballage, les contraintes de placement, et les contraintes de routage; les contraintes d'intégralité restent dans le modèle. On note alors L l'ensemble des contraintes relâchées, à savoir les contraintes d'isolement, les inégalités de capacité et les contraintes de latence et. Pendant le processus de branchement, la routine *Lazy Constraints* (disponible dans la procédure de rappel du solveur MILP [8]) est appelée chaque fois que le solveur trouve une nouvelle et meilleure solution entière: si la solution actuelle viole toute contrainte de L, il est ajouté en tant que *cut* au modèle réduit. Nous avons développé un cadre parallélisé dans lequel chaque processus parallèle est chargé de rechercher et d'ajouter les contraintes violées liées à un nœud ou un lien physique donné.

1.3.4 Sommaire

Dans cette section, nous avons présenté et discuté le problème de conception de *slices* dans les systèmes 5G et proposé une formulation MILP et des algorithmes exacts pour le résoudre. Des expériences numériques ont montré l'efficacité de chaque approche sur différentes classes d'instances. Par exemple, la rupture de symétrie proposée et les contraintes basées sur les limites inférieures ont conduit à une diminution importante de l'écart final : de 30% à 1% dans certains cas. De plus, notre algorithme Branch-and-Cut a pu réduire la taille de l'arbre Branch-and-Bound de 85% et a surpassé l'algorithme Branch-and-Bound du solveur dans tous les tests. Enfin, notre framework Row-Generation a surpassé l'approche Branch-and-Bound dans tous les tests, en particulier dans ceux avec des graphiques clairsemés.

1.4 Une math-heuristique pour le NSDP

L'objectif global de cette section est d'aller au-delà des travaux présentés dans la section précédente. À cette fin, nous proposons un cadre d'accès ouvert basé sur une heuristique mathématique pour le problème de conception de tranches de réseau présentant des résultats numériques pour évaluer l'efficacité de notre approche. L'idée globale de l'heuristique mathématique proposée repose sur la décomposition du NSDP en quelques sous-problèmes et sur leur résolution séquentielle. Ces sous-problèmes sont liés aux décisions suivantes : sélection fractionnée, regroupement NFS-NF, intégration de nœud NF et routage du trafic.

1.4.1 Description de l'algorithme

L'algorithme 1 présente le cadre global de notre approche. En entrée, l'heuristique reçoit une instance NSDP composée d'un graphe orienté G représentant le réseau physique avec l'ensemble de capacités C, un ensemble de requêtes de tranche S, dont chacune avec un ensemble de trafic demande $K(s)$, un ensemble F de types NFS et un ensemble N de fonctions virtuelles hôtes potentielles. En sortie, l'algorithme 1 renvoie un réseau virtuel à chaque demande de tranche $s \in S$ garantissant toutes les contraintes techniques imposées par les couches physiques et virtuelles.

Algorithm 1 : Math-heuristic for the NSDP

input : An NSDP instance $\mathbb{I}(G, S, F, N, C)$.
output: A solution to $\mathbb{I}$.

BestSolution, CurrentSolution $\leftarrow \emptyset$
while *a feasible solution to* $\mathbb{I}$ *is not found* **do**
  chooseCUs()
  foreach $s \in S$ **do**
    foreach $k \in K(s)$ **do**
      getPaths() ;
    choosePaths() ;
  selectSplit(s, CurrentSolution)
  while *a feasible embedding is not found* ***or*** *maximal number of tries is reached* **do**
    if $N \leftarrow$ *packNFSs() ;*
    is not feasible **then** stop and go to step 3;
    else if *embedNFs() fails and maximal number of tries is reached* **then** stop and go to step 3;
  if *routing() fails;*
  then stop and go to step 3 ;
  if *CurrentSolution is feasible and cost(CurrentSolution)* $<$ *cost(BestSolution)* **then**
    BestSolution $\leftarrow$ CurrentSolution
    if *rand()* $> \rho$ **then** try to find another solution to $\mathbb{I}$ by going to step 3;
return BestSolution

1.4.2 Sommaire

Dans cette section, nous avons présenté et discuté le problème de conception de tranches de réseau dans les systèmes 5G, en proposant un cadre d'accès ouvert basé sur une math-heuristique pour résoudre le problème d'optimisation sous-jacent. Des expériences numériques ont montré l'efficacité de notre approche sur différentes classes d'instances, qui pourraient atteindre des solutions quasi optimales dans un environnement d'exécution compétitif. En la comparant à une formulation de programmation linéaire à nombres entiers mixtes, la Math-Heuristique proposée pourrait réduire le temps d'exécution moyen et l'écart final jusqu'à 78% et 90%, respectivement. De plus, notre approche pourrait réduire la congestion sur le réseau physique, en équilibrant mieux le flux de données tout en tenant compte de toutes les contraintes techniques. Par exemple, la charge moyenne sur les liens physiques et les nœuds physiques pourrait être réduite de 16% et 35%, respectivement.

Sur une note pratique, vu que notre Math-Heuristique pourrait réduire la charge moyenne sur les nœuds physiques et les liens physiques, une extension difficile mais intéressante consiste à l'utiliser dans un algorithme en ligne. Notre approche pourrait potentiellement augmenter le taux d'acceptation des tranches, c'est-à-dire le rapport entre le nombre de tranches intégrées et

le nombre de requêtes, puisque la solution proposée par la Math-Heuristique distribue le flux de données entre plusieurs nœuds et liens, conduisant à une diminution de l'encombrement du réseau. Davantage de tests doivent donc être effectués pour évaluer les effets de l'heuristique mathématique proposée sur de tels scénarios et avant de pouvoir tirer des conclusions. Aussi, rappelons que, l'approche proposée étant une Math-Heuristique, l'optimalité des solutions trouvées par notre algorithme ne peut être assurée. Cependant, comme le montrent les résultats numériques présentés, l'efficacité de notre approche compense cet aspect et conduit à trouver des solutions avec un petit écart moyen sur des temps d'exécution relativement courts, même sur de grandes instances. Ainsi, il serait intéressant et très probablement très puissant de l'utiliser comme heuristique primale pour augmenter l'efficacité d'un algorithme exact.

1.5 Conception de slices avec des fonctions de réseau dédiéess

Dans cette section, nous présentons une autre variante du NSDP, où la politique d'isolation matérielle est toujours appliquée, c'est-à-dire que toutes les fonctions réseau sont dédiées aux tranches. Grâce à cette hypothèse, la complexité du problème étudié peut être diminuée, ce qui nous permet de proposer différentes approches heuristiques et mathématiques pour résoudre le problème d'optimisation associé. Cependant, les contraintes d'anti-affinité intra-tranche demeurent, ce qui représente la possibilité de regrouper deux NFS différents dans la même fonction de réseau desservant une *slice* donnée. La motivation pour conserver la cartographie NFS-NF vient de la diminution de la complexité opérationnelle liée au déploiement de chaque NS. En d'autres termes, l'installation de quelques fonctions réseau au lieu de plusieurs services de fonctions réseau permet aux opérateurs d'agréger dans une même entité tous les NFS partageant le même protocole de communication ou d'autres types d'affinité. Par conséquent, l'étape de configuration du déploiement de *slices* pourrait être potentiellement plus facile et plus rapide, réduisant ainsi les coûts d'exploitation [9]. Nous appelons cette variante *Network Slice Design with Dedicated Network Functions* (NSDP-DNF).

Dans ce qui suit, nous présentons différentes stratégies afin de résoudre efficacement les problèmes d'optimisation liés au NSDP-DNF. Tout d'abord, nous proposons une formulation compacte et une formulation étendue. Pour résoudre le premier, nous avons proposé une heuristique Relax-and-Fix qui repose sur la résolution répétitive de l'ILP lié proposé avec seulement

quelques variables entières et la correction ou la relaxation de la plupart des entiers et binaires restants. D'autre part, pour traiter le nombre exponentiel de variables dans la formulation étendue, un cadre basé sur la génération de colonnes est ensuite proposé.

1.5.0.1 L'algorithme Relax-and-fix

L'algorithme 2 présente le cadre global de notre approche. L'idée globale de l'heuristique proposée, ci-après dénommée Relax-and-Fix, repose sur la résolution répétitive de l'ILP proposé avec seulement quelques variables entières et sur la fixation ou la relaxation de la plupart des variables entières et binaires restantes. En entrée, l'heuristique reçoit une instance NSDP-DNF composée d'un graphe orienté G représentant le réseau physique avec l'ensemble des capacités C, un ensemble de demandes de tranches S, chacune avec un ensemble de demandes de trafic $K(s)$, un ensemble F de types NFS, un ensemble N de fonctions virtuelles hôtes potentielles et la stratégie de rythme $\rho \in \mathbb{Z}_+$. En sortie, il renvoie un réseau virtuel à chaque demande de tranche $s \in S$ garantissant toutes les contraintes techniques imposées par les couches physiques et virtuelles.

1.5.0.2 Algorithme basé sur la génération de colonnes

Nous décrivons maintenant le cadre de génération de colonnes pour résoudre le NSDP, qui repose sur la formulation E-ILP proposée comme problème principal. Nous avons d'abord introduit la formulation duale associée : les contraintes d'intégralité sont alors relâchées et deviennent donc respectivement $\pi_g^s \geq 0$. Ensuite, nous présentons une formulation ILP correspondant au sous-problème de tarification lié à chaque demande de tranche. Le problème du maître restreint Puisque $G(s)$ a un nombre exponentiel de modèles potentiellement appropriés pour chaque demande de tranche, la formulation E-ILP a un nombre exponentiel de variables. Par conséquent, nous proposons d'initialiser le problème maître avec seulement un sous-ensemble de colonnes (c'est-à-dire des variables liées à chaque $g \in G'(s) \subseteq G(s)$ pour chaque $s \in S$). Ce modèle est alors appelé *Restricted Master Problem* (RMP) et peut être généré en proposant des modèles sans exigence de capacité physique et de coût avec une valeur suffisamment grande (par exemple, $\mu_g^s = 10^5$). Nous proposons également d'initialiser le RMP avec la solution finale trouvée par l'algorithme Relax-and-Fix proposé décrit dans les sous-sections précédentes.

Algorithm 2 : Relax-and-Fix

input : An NSDP-DNF instance $\mathbb{I}(G, S, F, N, C)$ and a pacing strategy ρ.
output: A solution (if there exists one) to $\mathbb{I}$.

$N^* \leftarrow \{n_1\}$
Create model $\mathbb{M}$;
Relax all integrality constraints in $\mathbb{M}$
Order S ;
p $\leftarrow \rho$;
Let S^* be the set of embedded NSs: set it to a set
while $S^* \neq S$ ***and*** *feasibility condition is respected* **do**
    Set IntSlices to the first ρ slices in $S \backslash S^*$
    foreach *s in IntSlices* **do**
        enforce integrality on all related variables in $\mathbb{M}$
    solve $\mathbb{M}$
    if *a feasible solution is found* **then**
        foreach *s in IntSlices* **do**
            Fix values on all related variables in $\mathbb{M}$
        Add IntSlices to S^*
        p $\leftarrow \rho$
    else if $S^* \neq \emptyset$ **then**
        Remove the last embedded slice from S^*
        p←p+1;
    else
        Stop: no feasible solution exists
if $\mathbb{M}$ *is feasible* **then**
    PackNFSs() ;
    return the complete solution to $\mathbb{I}$
else
    return no solution

1.5.1 Sommaire

Dans cette section, nous avons présenté une autre variante du NSDP, appelée Network Slice Design with Dedicated Network Functions, où la politique d'isolation matérielle est toujours appliquée, c'est-à-dire que toutes les fonctions réseau sont dédiées aux tranches, en gardant cependant l'anti-tranche intra-tranche. -les contraintes d'affinité, qui représentent la possibilité de regrouper deux NFS différents dans la même fonction de réseau desservant une *slice* donnée. La motivation pour conserver la cartographie NFS-NF vient de la diminution de la complexité opérationnelle liée au déploiement de chaque NS. Nous avons ensuite présenté différentes stratégies afin de résoudre efficacement les problèmes d'optimisation liés au NSDP-DNF. Dans un premier temps, nous avons proposé une formulation compacte et une autre étendue. Pour résoudre le premier, nous avons proposé une heuristique Relax-and-Fix qui repose sur la résolution répétitive de l'ILP connexe proposé avec seulement quelques variables entières et en fixant

ou en relaxant la plupart des entiers et binaires restants. D'autre part, pour traiter le nombre exponentiel de variables dans la formulation étendue, un cadre basé sur la génération de colonnes a ensuite été proposé.

Dans nos simulations appliquant l'algorithme RF proposé, par exemple, l'écart moyen pourrait être réduit de 82,8% à 4,31% (resp. 48,4% à 5,77%) sur des instances moyennes-grandes (resp. moyennes) par exemple. De plus, bien qu'ILP n'ait pu trouver de solution entière réalisable en 1 heure d'exécution pour une instance de grande taille, la durée d'exécution moyenne et l'écart final étaient respectivement de 907 secondes et de 5,2% lorsque Relax-and-Fix était appliqué sur les mêmes instances. En ce qui concerne le cadre basé sur la génération de colonnes, cette approche pourrait améliorer l'écart dans toutes les tailles d'instances, en particulier lorsque la solution trouvée par l'algorithme RF a été donnée comme point de départ. Par exemple, l'écart et le temps d'exécution pourraient être réduits de 96%.

1.6 Conclusions finales

Nous avons défini et étudié le concept de communications appareil à appareil et de découpage de réseau dans les systèmes 5G et proposons des modèles mathématiques et des algorithmes innovants pour résoudre les problèmes d'optimisation sous-jacents. En particulier, nous avons étudié le problème de création de domaine, qui est un problème de routage et d'affectation de ressources survenant dans les futurs réseaux 5G. Nous avons proposé deux algorithmes, un exact et un heuristique, pour le résoudre. L'approche exacte est basée sur une formulation ILP nœud-arc renforcée par deux familles d'inégalités valides qui sont utilisées dans un cadre de branchement et de coupure. Nous avons également présenté une méthode de résolution basée sur une décomposition du DCP en deux sous-problèmes : le sous-problème de routage et le sous-problème d'allocation de ressources. Premièrement, un impact significatif des coupes dans le renforcement de la relaxation LP et la réduction du temps de calcul a été observé. Malgré un temps d'exécution plus long pour certaines instances utilisant les coupes proposées, nous avons pu observer que l'écart entre la solution racine et la solution finale est toujours plus petit lorsque des coupes de voisinage sont appliquées. En utilisant les réductions de capacité, cette amélioration est observée pour la moitié des instances considérées. Cependant, la taille finale de l'arbre après avoir trouvé la solution finale à l'aide de ces coupes peut être jusqu'à 76,40% plus

petite. Nos expériences ont également montré que l'approche heuristique proposée fonctionne bien, même sur de grandes instances avec jusqu'à 2 100 appareils et 1 500 demandes de service sur des réseaux à 7 cellules. Il serait intéressant et très probablement très puissant de l'utiliser comme heuristique primale pour augmenter l'efficacité d'un algorithme exact.

Nous avons également défini et modélisé l'approvisionnement des *slices* en tant que problème d'optimisation comprenant de nouvelles exigences de mappage et d'approvisionnement. En particulier, nous avons pris en compte les nouvelles dimensions de mappage apparaissant avec les systèmes 5G, modélisant la relation entre la séparation fonctionnelle flexible de l'accès radio, la séparation des fonctions du plan de contrôle et du plan de données et les politiques de partage. Différentes variantes du problème ont également été considérées et les modèles proposés sont conformes aux normes de fonctionnement. Nous avons démontré par des analyses numériques l'impact d'une prise en compte totale et partielle des contraintes particulières découlant des normes. Par exemple, nous avons rapporté des résultats numériques montrant que le fractionnement flexible apparaît comme un facteur clé pour faire face aux exigences hétérogènes de déploiement de services de communication distincts, entraînant une diminution considérable des coûts de *slice*. Dans nos simulations, le nombre de NFS nécessaires pour déployer les réseaux virtuels pourrait être réduit jusqu'à 56% selon laquelle des six politiques de partage proposées est appliquée à chaque *slice*. Nous avons également observé que différentes variantes liées au fractionnement flexible ont un impact important sur le réseau physique ; selon l'approche choisie, la charge moyenne sur les liens physiques pourrait être réduite d'un facteur 3.

Afin de renforcer la relaxation linéaire du MILP proposé, nous avons présenté plusieurs classes d'inéquations valides et les avons intégrées dans un cadre Branch-and-Cut pour résoudre le problème. Nous présentons en outre plusieurs stratégies pour réduire les symétries et la taille du modèle. Des expériences numériques ont montré l'efficacité de chaque approche sur différentes classes d'instances. Par exemple, la rupture de symétrie proposée et les contraintes basées sur les limites inférieures ont conduit à une diminution importante de l'écart final : de 30% à 1% dans certains cas. De plus, notre algorithme Branch-and-Cut a pu réduire la taille de l'arbre Branch-and-Bound de 85% et a surpassé l'algorithme Branch-and-Bound du solveur dans tous les tests. Enfin, notre framework Row-Generation a surpassé l'approche Branch-and-Bound dans tous les tests, en particulier dans ceux avec des graphiques clairsemés.

De plus, pour répondre à la complexité temporelle liée aux approches exactes proposées, nous avons présenté un cadre d'accès ouvert basé sur une heuristique mathématique pour résoudre le problème d'optimisation sous-jacent. L'idée générale de l'approche proposée reposait sur la décomposition du NSDP en plusieurs sous-problèmes et sur leur résolution séquentielle tout en englobant la séparation du plan de contrôle et du plan de données et de nouvelles dimensions de mappage et de décomposition influençant le placement et l'interconnexion des tranches. Des expériences numériques ont montré l'efficacité de notre approche sur différentes classes d'instances, qui pourraient atteindre des solutions quasi optimales dans un environnement d'exécution compétitif. En la comparant à une formulation de programmation linéaire à nombres entiers mixtes, la Math-Heuristique proposée pourrait réduire le temps d'exécution moyen et l'écart final jusqu'à 78% et 90%, respectivement. De plus, notre approche pourrait réduire la congestion sur le réseau physique, en équilibrant mieux le flux de données tout en tenant compte de toutes les contraintes techniques. Par exemple, la charge moyenne sur les liens physiques et les nœuds physiques pourrait être réduite de 16% et 35%, respectivement.

Nous avons également étudié la variante NSDP-DNF, où la politique d'isolement dur est toujours appliquée, c'est-à-dire que toutes les fonctions réseau sont dédiées aux tranches, en garantissant toutefois les contraintes d'anti-affinité intra-tranches, qui représentent la possibilité de regrouper deux NFS différents dans la même fonction de réseau desservant une *slice* donnée. Nous avons ensuite proposé différentes stratégies afin de résoudre efficacement le problème d'optimisation. Dans un premier temps, nous avons présenté une formulation compacte et une autre étendue. Pour résoudre le premier, nous avons proposé une heuristique Relax-and-Fix qui repose sur la résolution répétitive de l'ILP connexe proposé avec seulement quelques variables entières et en fixant ou en relaxant la plupart des entiers et binaires restants. D'autre part, pour traiter le nombre exponentiel de variables dans la formulation étendue, un cadre basé sur la génération de colonnes est ensuite proposé. Dans nos simulations appliquant l'approche Relax-and-Fix proposée, par exemple, l'écart moyen pourrait être réduit de 82,8% à 4,31% (resp. 48,4% à 5,77%) sur moyen-grand (resp. moyen) instances. De plus, bien qu'ILP n'ait pu trouver de solution entière réalisable en 1 heure d'exécution pour une instance de grande taille, la durée d'exécution moyenne et l'écart final étaient respectivement de 907 secondes et de 5,2% lorsque Relax-and-Fix était appliqué sur les mêmes instances. En ce qui concerne le cadre basé

sur la génération de colonnes, cette approche pourrait améliorer l'écart dans toutes les tailles d'instances, en particulier lorsque la solution trouvée par l'algorithme RF a été donnée comme point de départ. Par exemple, l'écart et le temps d'exécution pourraient être réduits de 96%.

Bibliography

[1] S. Abdelwahab, B. Hamdaoui, M. Guizani, and T. Znati, "Network function virtualization in 5G," *IEEE Comm. Mag.*, vol. 54, no. 4, pp. 84–91, 2016.

[2] T. Chen *et al.*, "Software defined mobile networks: concept, survey, and research directions," *IEEE Comm. Mag.*, vol. 53, no. 11, pp. 126–133.

[3] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5G: Survey and challenges," *IEEE Comm. Mag.*, vol. 55, no. 5, pp. 94–100, 2017.

[4] 3rd Generation Partnership Project, "3GPP TS 23.214 V14.0.0: Architecture enhancements for control and user plane separation of EPC nodes (Release 14)," *Technical Specification Group Services and System Aspects*, 2016.

[5] ——, "3GPP TR 28.801 V15.1.0: Telecommunication management; Study on management and orchestration of network slicing for next generation (Release 15)," *Technical Specification Group Services and System Aspects*, 2018.

[6] ——, "3GPP TS 23.501 V15.4.0: System Architecture for the 5G System (Release 15)," *Technical Specification Group Services and System Aspects*, 2018.

[7] L. Ford and D. Fulkerson, "«maximal flow through a network», canadian journal of mathematics," 1956.

[8] JuMP, *Julia for Mathematical Programming: Callbacks*, 2020, [Online]. Available on: https://jump.dev/JuMP.jl/v0.21.1/callbacks. Accessed: August 24, 2020.

[9] F. Malandrino, C. F. Chiasserini, G. Einziger, and G. Scalosub, "Reducing service deployment cost through vnf sharing," *IEEE/ACM Transactions on Networking*, vol. 27, no. 6, pp. 2363–2376, 2019.

le cnam

Wesley DA SILVA COELHO

Modeling and Optimization of 5G Network Design

HESAM UNIVERSITÉ

Résumé : Chaque seconde, une grande quantité de données numériques est transportée à travers les réseaux cellulaires du monde entier, et les attentes sont à une croissance considérablement accélérée avec des demandes de plus en plus importantes. En quelques années, ces réseaux pourraient atteindre leurs capacités maximales en termes de transmission de données. Pour faire face ces défis, Network Slicing a été présenté comme une nouvelle infrastructure virtualisée pour le système de réseau mobile de nouvelle génération. Cette technologie couvre désormais non seulement le niveau des applications, mais également la virtualisation des couches physiques et de commutation, avec différentes technologies d'accès radio. Ainsi, chaque fournisseur de services doit pouvoir déployer ses services sur des réseaux logiques, appelés Network Slices, spécifiquement adaptés à ses exigences techniques. Le mode de communication Device-to-Device est une autre approche présentée comme une alternative prometteuse à la communication traditionnelle dans les réseaux cellulaires. Cette technologie permet de réutiliser les ressources radio et de diminuer la latence de bout en bout des communications locales. Par conséquent, l'optimisation des ressources physiques dans les réseaux cellulaires devient cruciale pour mieux dimensionner et déployer les réseaux virtuels. L'objectif global de ce travail est donc de définir et d'étudier le concept de Device-to-Device Communication et Network Slice Design dans les systèmes 5G, en proposant des modèles mathématiques et des algorithmes innovants pour résoudre les problèmes d'optimisation sous-jacents.

Mots clés: Optimisation, Conception de réseau, Découpage réseau en tranches, Communication d'appareil à appareil

Abstract : Every second, a large amount of digital data is transported through cellular networks worldwide, and expectations are at a greatly accelerated growth with increasingly large requests. In few years, these networks could thereby reach their maximum capacities in terms of data transmission. To face these challenges, Network Slicing has been presented as a novel virtualized infrastructure for the new generation cellular network system. This technology now not only covers application-level abstraction but also physical and switching layers virtualization, with different radio access and link communication technologies. Hence, each service provider is to be able to deploy its communication services on top of logical networks, named Network Slices, specifically tailored to its technical requirements. The Device-to-Device communication mode is another approach presented as a promising alternative to traditional communication in cellular networks. This technology allows to reuse radio resources and to decrease the end-to-end latency of local communications. Consequently, the optimization of physical resources in cellular networks becomes crucial to better deploy virtual networks. The overall objective of this work is therefore to define and study the concept of device-to-device communication and network slice design in 5G systems, and propose mathematical models and innovative algorithms to solve the underlying optimization problems.

Keywords: Optimization, Network Design, Network Slicing, Device-to-Device