# A Math-Heuristic for Network Slice Design

Wesley da Silva Coelho
*CNAM, Orange Labs*
Chatillon, France
wesley.dasilvacoelho@orange.com

Amal Benhamiche
*Orange Labs*
Chatillon, France
amal.benhamiche@orange.com

Nancy Perrot
*Orange Labs*
Chatillon, France
nancy.perrot@orange.com

Stefano Secci
*CNAM*
Paris, France
stefano.secci@cnam.fr

***Abstract*—In this paper, we address the Network Slice Design problem, which arises from blueprinting end-to-end communication networks using fifth-generation (5G) radio access technology. With regard to new sharing policies and radio-access integration, it shows peculiar requirements with respect to conventional function placement and routing problems. To address the underlying optimization problem, we propose an open-access framework based on a math-heuristic that encompasses control-plane and data-plane separation and novel mapping and decomposition dimensions influencing the placement and interconnection of slices. Our framework also incorporates flexible functional splitting, with possibly different splitting for different slices while taking into consideration dependency factors such as varying network latency and data volume throughout the virtual access networks. Numerical results are then presented to assess the efficiency of our approach.**

***Index Terms*—Network slicing, flexible splitting, math-heuristic**

## I. Introduction

5G mobile systems integrate into a single physical network a variety of communication services, each with specific requirements in terms of latency, availability, and bandwidth, for instance. In this context, each service might use a set of customized logical networks, named *Network Slices* (NSs), specifically tailored to its requirements. Enhanced by Network Function Virtualization [1] and Software-Defined Networking [2] techniques, a network slice is composed of different virtualized *Network Functions* (NFs) representing different functionalities that used to be provided by dedicated machines. These virtual NFs are installed on physical servers, interconnected, and responsible for treating the data flow between two access points (data-plane NFs) and for managing and controlling the whole network slice (control-plane NFs). While each slice has only one set of control planes NFs, one ordered data-plane NF chain must be available to treat the flow from each access point (there might potentially be several of them serving each slice).

Addressing end-to-end network slicing problems requires, however, considering different virtual and physical network topologies, each with specific orchestration rules and technical constraints. For instance, recent 5G specifications [3]–[5] come with novel mapping dimensions that will affect the placement and interconnection of slices and network functions: (i) a communication service can be delivered by multiple slices; and (ii) Network Functions can be decomposed into micro-functions called *Network Function Services* (NFSs). These new mapping requirements come with new technical constraints to guarantee coherent provisioning of each communication service, such as NF scaling and sharing policies. Furthermore, the Flexible Radio Access Network splitting [6] that appears as a key technique to increase network efficiency, will bring even more flexibility in the way to design 5G networks. Indeed, Radio Access Network Functions could be split so that some functionalities could be distributed on a set of Distributed Units (DU), corresponding to access points, while others could be installed in a Centralized Unit (CU), corresponding then to pools of data-center servers. Hence, network designers should define the best functional splitting taking into account delay and bandwidth constraints on each physical path connecting DUs and CUs while considering dependency factors such as varying network latency and data volume between each pair of functions [7].

Fig. 1 shows an example of how to design different slices and embedding them into a physical network. In this example, we consider 2 slices requests, 5 traffic demands (e.g., from slice request 2: $DU23 \rightarrow App16$), 7 NFS types (3 for data-plane and 4 for control-plane), 8 NFs, and a physical network with 23 nodes (6 DUs, 12 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 slice request 1 are installed locally, 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 ($DU3 \rightarrow App7$) of slice 1, its virtual DP flow

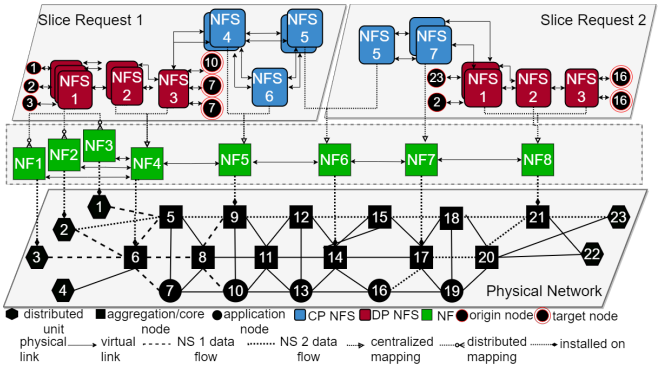


Fig. 1: Example of a solution for an NSDP instance.

This work is supported by the french Agence Nationale de la Recherche (ANR), Project MAESTRO-5G ANR-18-CE25-0012.

ISBN 978-3-903176-43-0 © 2021 IFIP

corresponds to the virtual link (NF1, NF4), and the related physical path $(3,6,7)$. In our previous work [8], 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%.

### A. Related Work

Optimization approaches related to network slicing problems mostly consider them either as Virtual Network Embedding (VNE) [9] or Function Placement and Routing (FPR) [10] problems. The former problem can be defined as follows: given a set of $r$ requested virtual network represented by graphs $G_v = (V_v, A_v)$, $v \in \{1,\dots,r\}$ 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 $u \in V_v$, $v \in \{1,\dots,r\}$ is mapped to a physical node in $V_p$, 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. [11] 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 [12] an online heuristic to address the computational complexity of the studied problem. In the same context, Fendt et al. [13] 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. [14], 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.

### B. Our contribution

Even though different works partially cover the network slicing [15], [16] and related sub-problems, such as functional split mode selection [6], [17], network slicing with NF sharing [18]–[20], and network slicing with NF scaling [21]–[23], no attention has been given to address jointly all aforementioned aspects in designing network slices and allocating resources to them. The overall objective of this work is to go beyond our previous work in [8] defining, for the Network Slice Design Problem (NSDP), a new math-heuristic including *splitting*, *mapping* and *provisioning* constraints made in the formulation compliant with the specifications [3]–[5]. Finally, we present numerical results to assess the efficiency of our approach.

TABLE I: Main notation

| | |
|---|---|
| **Set** | |
| $V_p$ | Set of all nodes. |
| $V_p^{du}$ | Set of all access nodes. |
| $V_p^{ac}$ | Set of all non-access nodes. |
| $V_p^{ap}$ | Set of all applications server nodes. |
| $A_p$ | 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** | |
| $c_u^c$ | amount of available resource $c$ on node $u$. |
| $\mu_u^c$ | 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 $g$. |
| $\lambda_f$ | compression coefficient of NFS $f$. |
| $\alpha_f^s$ | equals to 1 if a NFS type $f$ must be present in slice $s$; 0 otherwise. |
| $q_{fg}^{st}$ | equals to 1 if slice request $s$ 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 request $s$ admits sharing physical node with slice $t$; 0 otherwise. |
| $\eta_s$ | 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 $k$. |

## II. 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 in terms of graphs. Table I summarizes the notations.

*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 $V_p^{du}$, $V_p^{ac}$, and $V_p^{ap}$ 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 denoted by $C = \{c_u^1,\dots,c_u^c\}$, corresponding to the types of physical resources, 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 $d_a$.

*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 $f \in F^d$, $\lambda_f$ denotes a compression coefficient applied on the data-plane traffic of NFS $f$. We set $\lambda_{f_0} = 1$ and all the aforementioned parameters are set to 0 for the dummy NFS $f_0$.

*Network Slice Requests:* We denote by $S$ the set of network slice requests. Every request $s \in S$ is associated with a binary parameter $\alpha_f^s$ 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, generated by the slice and 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 $n_s$.

We define the target optimization problem as follows.

*Definition 2.1 (Network Slice Design Problem):* 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 decide whether they are to be installed centrally or distributed (i.e., 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 cost of deploying the slice requests and satisfying the technical constraints imposed by both physical and virtual layers.

## III. Proposed heuristic resolution approach

Algorithm 1 presents the global framework of our approach. The overall idea of the proposed math-heuristic[1] relies on decomposing the NSDP into several sub-problems to sequentially solve them. These sub-problems are related to the following decisions: split selection, NFS-NF packing, NF-node embedding, and traffic routing. As input, the math-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, it returns a virtual network for each slice request $s \in S$ ensuring technical constraints imposed by both physical and virtual layers.

[1]Mathematical programming techniques are applied within the framework.

**Algorithm 1:** 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() ; /* By Yen's algorithm */
7  choosePaths() ; /* See ILP (3)-(7) */
8  selectSplit()
9  **while** *a feasible embedding is not found* ***or*** *maximal number of tries is reached* **do**
10  **if** $N \leftarrow$ *packNFSs() ;* /* See Algorithm 2 */
11  *is not feasible* **then** stop and go to step 3;
12  **else if** *embedNFs() fails* ***or*** *maximal number of tries is reached* **then** stop and go to step 3;
13  **if** *routing() fails;* /* See Algorithm 3 */
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**
18  try to find another solution to $\mathbb{I}$ by going to step 3
19  **else**
20  **return** BestSolution

### A. 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 $\alpha$ is given by Equation (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.

$$\alpha = \sum_{f \in F^c} max\{c_f^c / c_u^c : c \in C, u \in V^{ac}\} \left\lceil \sum_{s \in S} \frac{n_s b_f}{cap(f)} \right\rceil + \sum_{f \in F^d} max\{c_f^c / c_u^c : c \in C, u \in V^{ac}\} \left\lceil \sum_{k \in K(s): s \in S} \frac{\lambda_{f-1} b^k}{cap(f)} \right\rceil \quad (1)$$

Then, the procedure builds a set $V^h$ of $\alpha$ 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 (2), which verifies the distance $dist(u,v)$, in terms of latency, between the node $u$ to all other nodes in $G$.

$$cent(u) = \frac{1}{\sum_{v \in V : v \neq u} dist(u,v)} \qquad ,\forall u \in V \qquad (2)$$

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 $\theta$ 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 [24] 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 $\pi_{uv}$ 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:

$$max \quad \sum_{u,v \in V^h | u \neq v} \pi_{uv} z_{uv} \qquad (3)$$

$$\sum_{p \in P(k)} x_p^k = 1 \qquad ,\forall k \in K(s) : s \in S \qquad (4)$$

$$\sum_{k \in K(s)} \sum_{p \in P(k)} \lambda_{uv}^p x_p^k = z_{uv} \qquad ,\forall u,v \in V^h | u \neq v \qquad (5)$$

$$x_p^k \in \{0,1\} \qquad ,\forall k \in K(s) | s \in S, \forall p \in P(k) \qquad (6)$$

$$z_{uv} \in \mathbb{N}_0 \qquad ,\forall u,v \in V^h | u \neq v \qquad (7)$$

Since the aim is to share ordered data-plane NFS chains among as many traffic demands as possible, the objective function (3) consists in maximizing the number of chosen paths that have similar structures. For this end, Equation (5) calculates how many activated paths pass by node $u \in V^h$ before node $v \in V^h$, where $\lambda_{uv}^p$ 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 (4) ensures that there is exactly one path selected to each traffic demand $k$, (6) and (7) are the variable domain constraints. The proposed formulation is then solved by a distributed parallel branch-and-bound algorithm [25] using multiple threads to solve each singular branch-and-bound tree's node.

**Algorithm 2:** 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'
```

In what follows, 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.

### *B. Network Function Service Packing*

In what follows, *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 [26]. 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 $(q^{ss'}_{ff'} q^{s's}_{f'f} = 0) \vee (u \neq u')$ holds. 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 maximal clique size is calculated with *getCliquesSize()* on step 2 by applying the Grimmett-McDiarmids greedy algorithm [27]: 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 maximal 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 [28]. 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 $w^s_f \in \mathbb{R}_+$ 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 (8).

$$w^s_f = \begin{cases} \dfrac{n_s b_f}{cap(f)} & \text{if } f \in F^c; \\ \dfrac{\sum\limits_{k \in K(s)} \lambda_{f-1} b_k}{cap(f)} & \text{if } f \in F^d. \end{cases} \quad ,\forall s \in S, \forall f \in F^{cent}(s) \qquad (8)$$

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 $(max\{\dfrac{c^c_f w^s_f + c^c_{f'} w^{s'}_{f'}}{c^c_u} : c \in C, u \in V^h\} > 1) \vee (q^{ss'}_{ff'} q^{s's}_{f'f} + q^{ss'} q^{s's} < 2)$ 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.

### C. 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.

### D. 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 3 depicts our approach to 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

**Algorithm 3:** 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** $\emptyset$

generated. For this purpose, Yen's algorithm [24] is used to find $\theta$ 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.

### E. Final Solution

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

$$cost(Solution) = \sum_{f \in F} \sum_{n \in N} \sum_{u \in V} \sum_{c \in C} \mu_u^c y_{nu}^f \tag{9}$$

Where $y_{nu}^f \in \mathbb{Z}_0$ and $\mu_u^c$ 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 $\phi$. For instance, setting $\phi$ to 60, the probability $P(rand() \geq \rho)$ that $rand()$ is greater than $\rho$ 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.

### F. Algorithm's Time Complexity

Table II 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 I. Let us recall that, besides chooseCU() and embedNFs(), all procedures are able to be run in a distributed parallel way.

TABLE II: 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 III: Instance Sizes

| Instance size | $\|V\|$ | 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 IV: 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, which is set to 6 milleseconds. |
| 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 $q_{fg}^{st}$ are set to 0; they are randomly chosen. |
| Strong (S) | 75% of isolation parameters $q^{st}$ and $q_{fg}^{st}$ are set to 0; they are randomly chosen. |

## IV. Numerical Experiments

Let us first detail the simulation settings. We propose different instance sizes (see Table III), 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 IV, 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

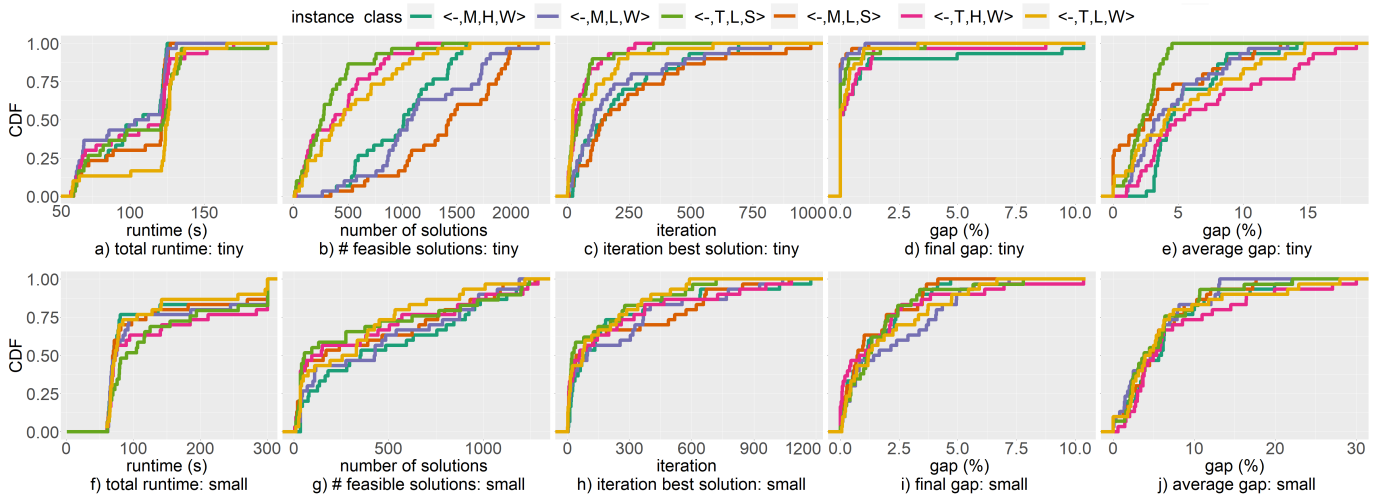


Fig. 2: Quantitative analyses: tiny and small instances.

name from tables III and IV. 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 ILOG 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 following the parameters in Tables III and IV. The data-set and the source code are available on [29].

### A. 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, hereafter referred to as to *MILP*, introduced in [8], whose objective function is replaced by the solution cost (9). Concerning step 17 of Algorithm 1, we set $\rho$ to $1-60/seconds$ for tiny and small instances and $1-600/seconds$ for all other sizes in our simulations.

Fig. 2 shows different results on tiny and small instances. First, as seen in Fig. 2a and Fig. 2f, 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. 2g), this behavior led to an increase in the number of solutions for tiny-size instances (see Fig. 2b). 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. 2c and Fig. 2h, 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. 2d and Fig. 2i 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. 2e and Fig. 2j, 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 V 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 IV) was randomly chosen. While the third and fourth

TABLE V: 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.

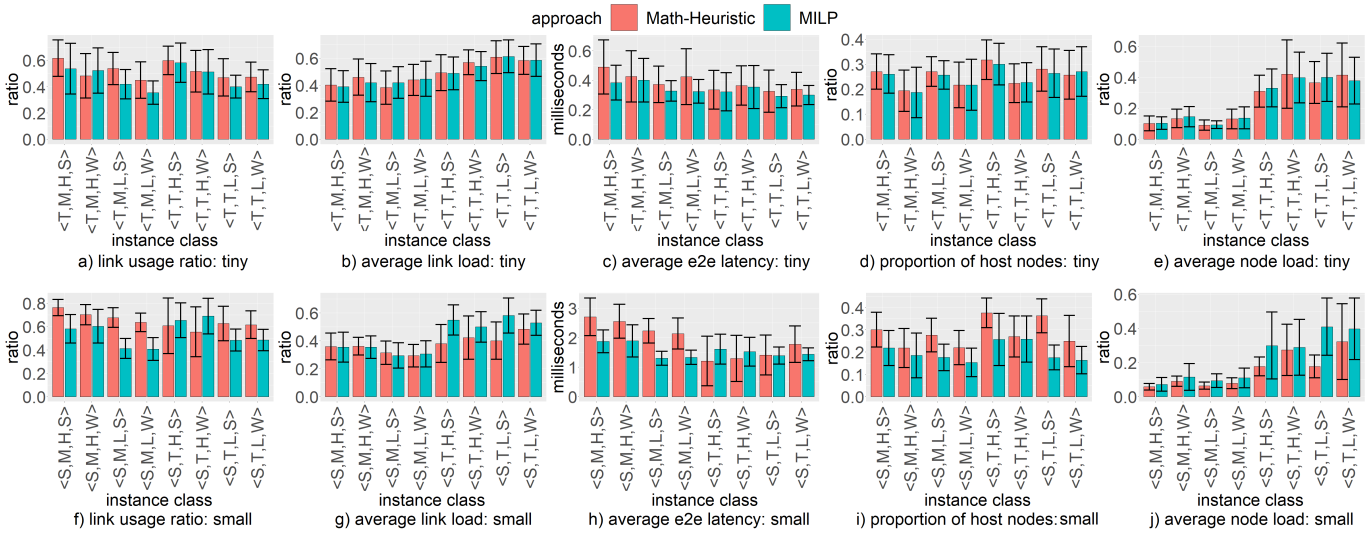


Fig. 3: Qualitative analyses: tiny and small instances.

columns of Table V 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 V, 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.

### *B. 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. 3 shows the impact of each approach on the aforementioned network parameters; the error bars correspond to the related 95% confidence interval. 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. 3a and Fig. 3f). However, the average load on active links remains the same on tiny instances (see Fig. 3b) and on small instances without strict capacity constraints (see Fig. 3g). Also, as seen in Fig. 3g, 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. 3c and Fig. 3h).

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.3i, 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 proportion 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.3d and Fig.3i). 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. 3j, 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. 3j); we did not observe an important impact on the average load of physical nodes on tiny instances (see Fig. 3e).

## V. Concluding Remarks

In this work, 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 subproblems 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 better 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 algorithm 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.

## References


[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, M. Matinmikko, X. Chen, X. Zhou, and P. Ahokangas, "Software defined mobile networks: concept, survey, and research directions," *IEEE Comm. Mag.*, vol. 53, no. 11, pp. 126–133.

[3] 3rd Generation Partnership Project, "3GPP TR 38.913 V14.3.0; Study on scenarios and requirements for next generation access technologies ," 2017.

[4] ——, "3GPP TR 28.801 V15.1.0: Study on management and orchestration of network slicing for next generation network ," 2018.

[5] ——, "3GPP TS 23.501 V15.4.0: System Architecture for the 5G System ," 2018.

[6] 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*.

[7] 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.

[8] 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.

[9] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Comm. Surveys & Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[10] 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.

[11] 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.

[12] ——, "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.

[13] 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.

[14] J. Liu, B. Zhao, M. Shao, Q. Yang, and G. Simon, "Provisioning optimization for determining and embedding 5g end-to-end information centric network slice," *IEEE Transactions on Network and Service Management*, 2020.

[15] 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.

[16] B. Tan, J. Wu, Y. Li, H. Cui, W. Yu, and C. W. Chen, "Analog coded softcast: A network slice design for multimedia broadcast/multicast," *IEEE Transactions on Multimedia*, vol. 19, no. 10, pp. 2293–2306, 2017.

[17] X. Wang, L. Wang, S. E. Elayoubi, A. Conte, B. Mukherjee, and C. Cavdar, "Centralize or distribute? a techno-economic study to design a low-cost cloud radio access network," in *ICC*. IEEE, 2017.

[18] F. Malandrino and C.-F. Chiasserini, "Getting the most out of your vnfs: Flexible assignment of service priorities in 5G," in *WoWMoM 2019*.

[19] 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.

[20] M. R. Crippa, P. Arnold, V. Friderikos, B. Gajic, C. Guerrero, O. Holland, I. L. Pavon, V. Sciancalepore, D. von Hugo, S. Wong *et al.*, "Resource sharing for a 5G multi-tenant and multi-service architecture," in *European Wireless Conference*. VDE, 2017.

[21] I. Alawe, Y. Hadjadj-Aoul, A. Ksentini, P. Bertin, and D. Darche, "On the scalability of 5G core network: the AMF case," in *CCNC*. IEEE, 2018, pp. 1–6.

[22] I. Alawe, Y. Hadjadj-Aoul, A. Ksentini, P. Bertin, C. Viho, and D. Darche, "Smart scaling of the 5G core network: an rnn-based approach," in *2018 GLOBECOM*. IEEE, 2018, pp. 1–6.

[23] 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.

[24] 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.

[25] 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.

[26] E. Malaguti and P. Toth, "A survey on vertex coloring problems," *International transactions in operational research*, vol. 17, no. 1, pp. 1–34, 2010.

[27] 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.

[28] M. M. Sysło, "Sequential coloring versus welsh-powell bound," *Discrete mathematics*, vol. 74, no. 1-2, pp. 241–243, 1989.

[29] W. da Silva Coelho. (2021) A Math-Heuristic for the NSDP: source code and instances. [Online]. Available: https://github.com/wdscoelho/NSDP_heuristic