# Meta-heuristics for placement and chaining of micro-services

Hichem Magnouche[1], Caroline Prodhon[1], Guillaume Doyen[2]

[1]ICD-LIST3N, University of Technology of Troyes, Troyes, France
[2]Imt-Atlantique, Rennes, France
{hichem.magnouche,caroline.prodhon}@utt.fr
guillaume.doyen@imt-atlantique.fr

***Keywords*** : Virtualisation of network functions, Optimisation, Orchestration, Low latency, Micro-services

## 1 Introduction

The virtualization of network architectures offers numerous advantages and is becoming an increasingly important concern in the field of telecommunications. It provides greater flexibility for the deployment of network functions (NF), allowing the implementation of emerging services such as those requiring low latency. In this area, the concepts of microservices and network programmability are promising solutions but raise issues regarding orchestration rules, namely placement and chaining. After reviewing optimization-related issues, we proposed a model for the placement and chaining of microservices as well as three cohabitation strategies between low latency SFC and Best Effort requiring less latency. Facing an extremely long resolution time, we propose, in this work, a heuristic allowing the placement and chaining of microservices in a relatively short time. This research is conducted within the framework of ANR MOSAICO project. It is important to note that the virtualization of network architectures is a rapidly evolving field, with new technologies and strategies constantly emerging. As such, ongoing research and development in this area is critical for ensuring that telecommunications networks remain flexible and responsive to the needs of modern businesses and consumers.

## 2 Microservices orchestration

After studying the state-of-the-art solutions in the field of orchestration (placement and chaining) of virtualized network functions, we proposed an approach in previous works [2, 3] that leverages the key characteristics of microservices, namely mutualization and parallelization, to deploy SFC with very low latency. Our approach consists of a preprocessing algorithm and a mathematical model (MILP) for placement and chaining. The preprocessing algorithm enables the mutualization of mutually shareable microservices within the same SFC, which reduces their length and consequently their latency. It also detects parallelizable microservices and creates a new chaining that allows them to operate in parallel. To determine whether two or more microservices can be mutualized or parallelized, the algorithm relies on tables from the state of the art [4, 6]. Regarding the MILP, it allows the placement and chaining of microservices while optimizing their parallelization. One of the key characteristics of microservices is their ability to execute in parallel, which reduces their execution latency. The objective function aims to minimize the number of SFC exceeding the required latency. Our proposed model produces optimal solutions after a few hours (using CPLEX) for instances of a few SFC on infrastructures consisting of 5-8 nodes.

# 3 Heuristic of placement and chaining of microservices

The evaluations carried out on the proposed approach demonstrate a significant gain in terms of latency compared to a monolithic approach. However, testing this model on larger instances in terms of nodes and SFC is impossible, as the associated resolution time grows exponentially. To address this issue, we have developed a heuristic method that, based on the parallelism characteristics of microservices, produces "correct" solutions in polynomial time.

The steps of the heuristic for each SFC are as follows : (1) Calculate the $k$ shortest paths between its source and destination, and (2) deploy the microservices that are relevant to them.

## 3.1 Computing the $k$ shortest paths

By analysing the literature review, we found that a large number of heuristics use Dijkstra algorithm to compute the shortest path between the source and the destination, as well as alternative shortest paths when a saturation occurs. [1], for example, remove the saturated node from the shortest path before running the Dijkstra algorithm again, while [5] remove the arc with the least latency before running Dijkstra again. This way of doing things allows for the traversal of a set of shortest paths without guaranteeing that they are indeed "the" shortest paths.

For the heuristic we developed, we decided to use the Eppstein algorithm to calculate the $k$ shortest paths. This approach guarentees that if we decide to use the $k^th$ shortest path, that is really the optimal $k^th$ one. The choice of the Eppstein algorithm was made because of its optimality as well as its complexity, which is in $O(m + nlog(n) + klog(k))$, where $m$ represents the number of arcs, $n$ the number of nodes, and finally $k$ the number of paths to be calculated.

## 3.2 Deployment of microservices.

The deployment of microservices is done dynamically, taking into account the network configuration in order to maximize the parallelization of microservices. In fact, for the deployment of each microservice, the heuristic checks the possibility of parallelization with the preceding microservices. If it can run with them and they are all deployed on the same node, they will therefore run in parallel. However, if they are not placed on the same node, the heuristic checks if the current node can contain them and moves the preceding microservices to it. This approach of parallelization during deployment thus allows for the adaptation of placement according to the abilities of the microservices to function in parallel.

# 4 Meta-heuristics for placement and chaining

The limit of our heuristic approach is that the SFC are processed one by one in a sequential manner, which penalizes the effective latency of the last SFC and provides a latency beyond what is necessary for the first ones. Therefore, we are working on a metaheuristic that relies on the solutions produced by the heuristic in order to improve them through local search.

The principle of our approach is to start from a solution generated by our constructive heuristic, and then to use destruction/reconstruction procedures. These respectively remove and reinsert micro-services from the current solution according to various strategies. A first "easy" approach would be to remove complete SFC, for example, by pair with a proba associated with latency or latency gap (e.g. the one with the highest and the one with the lowest latency) and then try to reinsert them. Depending on the results and time we will consider other local search strategies.

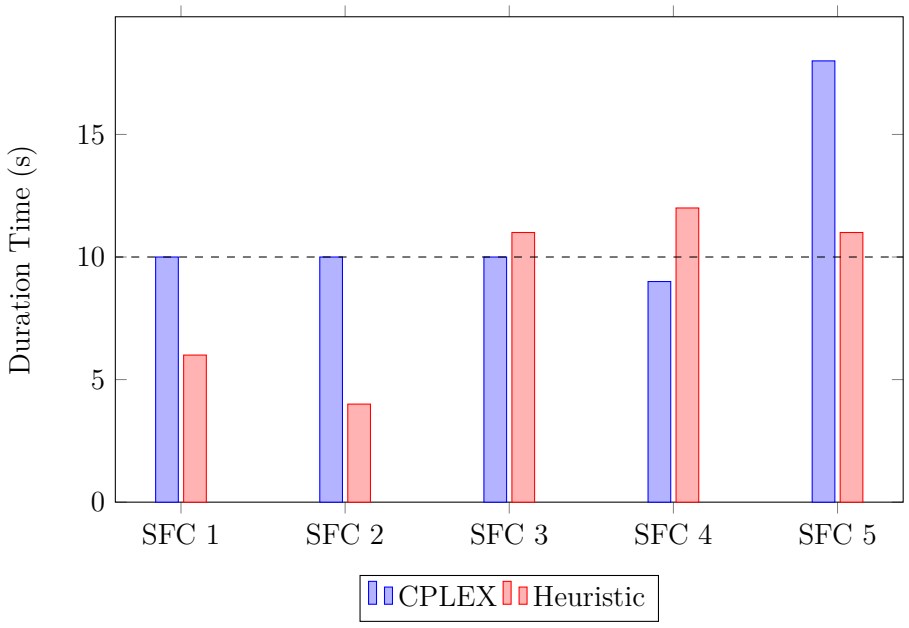


FIG. 1 – Effective latency of SFC under the exact (CPLEX) and heuristic approach

# 5 Analysis of results

## 5.1 Heuristic

We tested our heuristic on 9 scenarios in which we varied the number of nodes in the infrastructure {10,14,18}, the number of SFC 25,35,50, as well as the capacity of the infrastructure {110%, 130%, 150%} of the number of required microservices. In order to compare our heuristic, we used our mathematical model that produces optimal solutions.

The results we obtained proved the speed of our heuristic, which is 2300 times faster than the mathematical model. For the quality of the solution, we are on average at 1.26 of the optimal solution. It should be noted that the number of scenarios is small, which can have a significant impact on the average distance of the solution to the optimal solution.

The analysis also allowed us to detect a paradoxical behavior. Indeed, despite the fact that under the heuristic approach we have a larger number of SFC exceeding the required latency, the total average execution time of all SFC is smaller under the heuristic approach than under the exact approach. This is explained by the fact that under the mathematical approach, the SFC not respecting the latency will exceed the required latency by a large margin compared to the heuristic approach. Also, under the heuristic approach, the first SFC are always favored and the SFC not respecting the latency are always those placed at the end. On the 1 we tested 5 SFC under the exact approach (CPLEX) and the heuristic approach. All 5 SFC have a latency constraint of 10ms. We can see that CPLEX manages to respect the latency of 80% of the SFC contrary to the heuristic approach which only manages to respect 40%. But what is interesting to note is the fact that the only SFC which does not respect the latency under the exact approach, largely exceeds the required latency contrary to the exact approach where the whole of the SFC are more or less optimized. We also notice that the first 2 SFC on the heuristic approach are largely advantaged compared to the 3 remaining SFC.

## 5.2 Meta-heuristics

The results of our meta-heuristic will be presented during the presentation, as its development is still ongoing.

## Références


[1] Anish Hirwe and Kotaro Kataoka. Lightchain : A lightweight optimisation of vnf placement for service chaining in nfv. In *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, pages 33–37, 2016.

[2] Hichem Magnouche, Guillaume Doyen, and Caroline Prodhon. Leveraging micro-services for ultra-low latency : An optimization model for service function chains placement. In *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*, pages 198–206, 2022.

[3] Hichem Magnouche, Guillaume Doyen, and Caroline Prodhon. A fair approach for micro-services function chains placement in ultra-low latency services. In *IEEE TNSM*, 2023 (Under Revision).

[4] Zili Meng, Jun Bi, Haiping Wang, Chen Sun, and Hongxin Hu. Micronf : An efficient framework for enabling modularized service chains in nfv. *IEEE Journal on Selected Areas in Communications*, 37(8) :1851–1865, 2019.

[5] Krishnamoorthy Sivalingam, Akshay Gadre, and Anix Anbiah. Centralized approaches for virtual network function placement in sdn-enabled networks, 2018.

[6] Yang Zhang, Bilal Anwer, Vijay Gopalakrishnan, Bo Han, Joshua Reich, Aman Shaikh, and Zhi-Li Zhang. Parabox : Exploiting parallelism for virtual network functions in service chaining. In *SOSR*, pages 143–149. ACM, 2017.