A Knapsack-based Optimization Algorithm for VNF Placement and Chaining Problem

Abdeldjalil Ikhelef, Mohand Yazid Saidi, Shuopeng Li, Ken Chen

A Knapsack-based Optimization Algorithm for VNF Placement and Chaining Problem

Issam Abdeldjalil IKHELEF[1], Mohand Yazid SAIDI[1], Shuopeng LI[2] and Ken CHEN[1]
[1] *L2TI - Institut Galilée, Université Sorbonne Paris Nord, 93430 Villetaneuse, France*
[2] *Faculty of Information Technology, Beijing University of Technology, Beijing, China*
issam.ikhelef@edu.univ-paris13.fr, saidi@univ-paris13.fr, lishuopeng@bjut.edu.cn, ken.chen@univ-paris13.fr

***Abstract*—During the last decade, we are witnessing the emergence of NFV and SDN to reduce CAPEX and OPEX. Under the SDN paradigm and thanks to NFV, a service can be swiftly deployed by the chaining of several VNFs forming an SFC running on a virtualized infrastructure. Nowadays, there are still quite a number of issues related to SFCs, among them, the optimal placement of SFC components. In this paper, we focused on the variant of the resource allocation cost optimization problem of VNF placement and chaining for limited resources on the servers. After proving that the problem of VNF placement is NP-Hard and equivalent to the multiple knapsack problem, we proposed a genetic algorithm-based meta-heuristic to solve large instance of our VNF placement and chaining problem variant. Simulation results show that our genetic algorithms are efficient since they reduce the SFC mean cost and improve the accepted requests ratio.**

***Index Terms*—Virtual Network Function, Network Function Virtualization, Service Function Chain, Optimization, Multiple Knapsack Problem, Genetic Algorithm, Meta-heuristic.**

I. Introduction

NFV (Network Function Virtualization) is part of a radical change in the way hardware and software equipment are operated and interacted on the network. Together with SDN (Sofware-Defined Networking) paradigm, NFV creates a networking environment which is rich in programmable capabilities and automation. NFV replaces expensive dedicated hardware devices, such as routers, firewalls and load balancers, by their software equivalent, termed as virtual network functions (VNFs), that run as virtual devices on standards servers [1].

With NFV, service providers can run virtual network functions (VNFs) on different servers or move them as needed when demand changes. This flexibility allows them to accelerate the distribution of services and applications. For example, if a customer requests a new network function, the service provider can install it on an adequate place and later, it is no longer needed, remove it [2].

A network service is composed of different VNF forming an SFC (Service Function Chain). The latter is to be placed across the network using software provisioning [3]. Typical examples of network functions are: firewall, SSL (Secure Sockets Layer), load balancer, NAT (Network Address Translation), etc. Each SFC (see the bottom of Figure 1 for an example) specifies an ordered sequence of VNFs from an *source* till a *target*. With NFV, these functions (VNFs) can be placed at any adequate node in the network with respect to sequence order and resource requirements.

To provide an SFC, the related VNFs should be placed on VNF server nodes and chained through a route capable of accommodating the traffic from the *source* to the *target* of the SFC. Determining the optimal placement and chaining of VNFs, which consists of providing and accommodating SFCs, corresponds to the well-known problem of VNF placement and chaining (VNFPC) that is a NP-hard problem.

The objective of this paper is to determine the VNF placement and chaining that optimize the resources allocation cost when network resources are limited on the nodes and sufficiently abundant on the links. We obtain thus a particular version of the VNFPC problem with relaxation of link constraints, we term this problem as RVNFPC. The relaxation of the link constraints (by assuming enough resources on links) makes sense for networks/slices where links have large bandwidth capacities. Furthermore, we also can use this approach in a dynamic programming perspective by solving the VNFPC problem in two steps: (1st) determine a set of plausible VNF placement candidates without considering link constraints, then (2nd) determine a valid candidate from the results of the first step by finding a route satisfying the link constraints.

Contrary to most of the approaches proposed in the literature, we propose here to solve RVNFPC directly in its domain of definition without going through ILP. After proving that RVNFPC can be solved with a variant of the multiple knapsack problem, we will propose knapsack problem-based efficient heuristics to solve large instances of the RVNFPC problem.

The rest of this paper is organized as follows: in Section II, we review research contributions that tackle the VNFPC problem. In Section III, we describe the VNFPC problem in its general form and present an Integer Linear Programming (ILP) model that permits to obtain exact solutions. Then, we consider a relaxation of the problem (RVNFPC) and solved it with the genetic algorithms described in Section IV. Section V presents the simulation results and finally, conclusions are given in Section VI.

II. Related work

In recent years, the problem of VNF placement and chaining has received considerable attention from researchers.

Studies can be classified into two categories: the first one aims to optimize the resources or energy consumption and the second one focuses only on the optimization of overall costs.

A. Resources consumption optimization

Under this first category, authors in [4] formalized the form of SFC in NFV and Edge Computing (EC) enabled networks, and then formulated the VNF placement problem as an ILP model aiming to minimize the total resource consumption. They came up with a priority-based greedy solution which consists of a priority-based SFC mapping algorithm and VNF merge algorithm that gives priority to the SFC clusters with larger resource consumption. As indicated by the authors themselves, their method suffers from a scalability problem when the number of SFCs is big.

The work in [5] supports the idea of a centralized approaches for VNF placement in SDN-enabled networks. The authors do not consider resources costs in the optimization and focused only on the network congestion and reducing resources usage. To solve the Mixed Integer Linear Programming (MILP) model that they proposed, the authors used a Divide-and-conquer solution with a modified version of Dijkstra's algorithm. At each step of the divide-and-conquer algorithm, a VNF allocation is made and a new instance of the problem is spawned corresponding to the allocation.

The work in [6] tackles the VNFPC problem from the perspective of minimizing the consumption of network resources which only include bandwidth. They modelled the problem as an ILP with an objective function that calculates the total bandwidth consumed by all the requested flows. Furthermore, they studied the network resources consumption of various service-chaining strategies, the network itself may have a varying number of NFV-capable nodes and proprietary hardware solution. The ILP model in [6] suffers also from scalability.

To minimize the energy consumption, [7] proposed an ILP and two heuristics for online and batch calculations. Online heuristic selects a limited number of candidate hosts in the infrastructure to control complexity. It handles SFC requests sequentially. The batch heuristic works on a set of SFC requests. The weakness of the online algorithm is that VNF requests that cannot be placed are immediately rejected while stored with the batch algorithm until some resources are freed.

B. Overall cost optimization

In [8], the VNFPC problem was formulated as an ILP model to minimize the latency, Service-Level Objective (SLO) violation cost, hardware utilization, and VNF readjustment cost. To solve the model, the authors used k-medoids clustering approach that proactively partitions the substrate network into a set of disjoint clusters and appropriately eliminates some cost functions of the optimization problem to increase its feasibility in large-scale networks.

In [9], the VNFPC problem is studied from the perspective of minimizing both of allocation host resources and bandwidth costs. Authors of [9] proposed a MILP solved that they solved with a heuristic that provinging near-optimal solution in a reasonable time. Their heuristic uses an adjustment parameter which controls the trade-off between the speed and the precision of the solution.

The VNFPC problem has been addressed in [10] using an ILP based scheme to minimize the overall cost. The authors used an approach that consists of two heuristics called *SFC-Reactive* scheme and *SFC-Proactive* scheme. The *SFC-Reactive* scheme aims at fulfilling the scalable SFC requests without changing the Service Function Path (SFP) while the *SFC-Proactive* scheme is intended to optimize the SFP for better serving the subsequent arriving SFC requests and thus achieving better network performance.

Authors in [11] formulated the VNFPC problem as a decision tree to reduce significantly the complexity of SFC in clouds and increase provider revenue. Each node in the decision tree corresponds to a virtual resource embedding and each tree branch to the mapping of a client request in some physical candidate. They derived a new algorithm based on the Monte Carlo Tree Search (MCTS) to incrementally build and then search within the decision tree.

In [12], authors address the placement of SFC by finding the best hosts for the VNFs while respecting user requirements and maximizing provider revenue. [12] proposes a novel Eigen-decomposition-based approach for the placement of VNFs and PNFs (Physical Network Functions) chains in networks and cloud environments. It also presents a heuristic based on a custom greedy algorithm to compare performance and assess the capability of the Eigen-decomposition approach.

Because networks can be divided into several entities belonging to different service providers which are reluctant to reveal their internal topologies, the authors in [13] formalized an ILP model and proposed a heuristic that allows the NFV orchestrator to place the network function chains based only on an abstract view of the infrastructure network. They leveraged this approach to address the complexity of the problem in large mono- or multi-service providers networks.

The aim of [14] is to provide high performance and scalable algorithms capable of finding optimal solutions for the VNFPC. First, an exact algorithm based on Perfect 2-Matching theory is proposed to solve, in polynomial time, the case of SFC composed of up to 3 VNF based chains. Then, authors proposed an approach based on matrix analysis that combines matrix products with a simple linear program to find an optimal control of traffic flows on each placed VNF. Finally, they proposed a multistage graph-based approach that is built as a new extended multistage graph representing the servers available to host the required VNFs and their interconnections.

Most of the approaches described here derive their solutions from ILP which is only effective for small sizes of the problem. Instead of looking for heuristics that solve ILP, it would be wise to try to solve VNFPC problem directly in its definition domain, as we propose it in this paper. The advantage of such approach is its ability in terms of scalability.

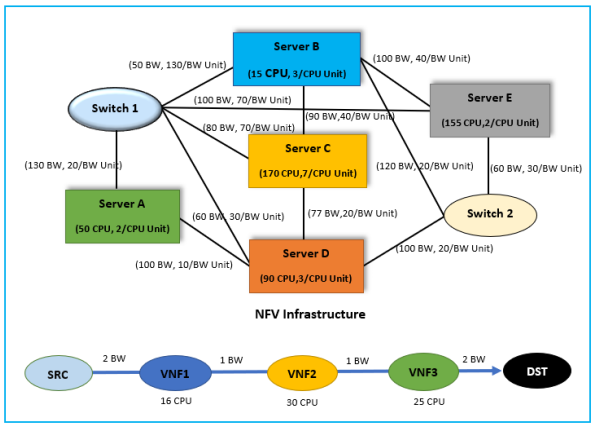


Fig. 1: An example of NFV-I and SFC topology.

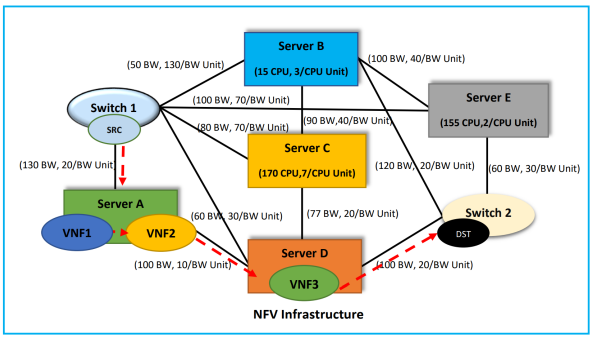


Fig. 2: Mapping of SFC in NFV-I.

III. Problem Description

In this section, we first present more in details the VNFPC problem. After explaining the context, we define and explain the impact and variables we used to formulate the problem as an ILP optimisation problem. Then, by relaxing the link constraints in the precedent problem we obtain a new problem (RVNFPC) that we transformed into a multiple knapsack problem. For the ease of understand, let us give an example.

The upper part of Figure 1 represents a NFV-Infrastructure (NFV-I) composed of 5 node servers (*A*, *B*, *C*, *D* and *E*) and two switches (*Switch1* and *Switch2*) representing symbolically the source and target of the SFC flow. Each node is illustrated with a different color and has a certain amount of capacity to host VNFs and its own pricing policy for VNFs hosting.

The nodes or servers are interconnected by substrate links with given costs and high resource capacities. The substrate links correspond to virtual links embedded on substrate paths that interconnect the extremity nodes supporting the servers.

The bottom of Figure 1 shows an SFC with 3 VNFs to be deployed in the NFV-I. Each VNF requires a number of CPUs. We assume that *VNF1* requires 16 CPUs, 30 CPUs for *VNF2* and 25 CPUs for *VNF3*.

The optimal placement and chaining of the SFC in Figure 1 is highlighted by the dashed red lines in Figure 2: *VNF1* and *VNF2* are placed on server *A* whereas *VNF3* is placed on server *D*. The total cost of the solution is therefore equal to 257 units (20×2 + 2×16 + 0×1 + 2×30 + 10×1 + 3×25 + 20×2).

A. Mathematical Modelling of VNFPC problem

The substrate network NFV-I is modeled by an undirected weighted graph, noted $G_s = (V_s, E_s)$ where V_s is the set of nodes (vertices) corresponding to NVF-I servers and switches, and E_s is the set of virtual links (edges).

Each node is associated with a list of its resources : CPU, memory, storage, etc. Each type of resources is identified by its index number $r \in R = \{0, 1, 2, ...\}$.

For a given resource r of a node u, its costs and capacity (initial allocation) are denoted by respectively α_u^r and C_u^r. Similarly, for each virtual link $l_s = (l_s^a, l_s^b)$, its capacity is denoted by CB_{l_s}, and its cost by β_{l_s}.

Each SFC is also modeled as a graph interconnecting a *source* (represented by u_s, the switch on its side), k VNFs and a *target* (represented by u_t, the switch on its side), as shown in Figure 1. A SFC is represented by a graph $G_f(u_s, u_t) = (V_f, E_f)$, where E_f is the set of virtual links which connect the VNFs and V_f includes all the VNFs of the service chain plus B_s and B_t which represent symbolically on V_f the two switches u_s and u_t, respectively.

Each VNF $V \in V_f$, is characterized by a demand for resource $r \in R$ denoted D_V^r. Each virtual link $l_V \in E_f$ is characterized by a demand for bandwidth denoted DB_{l_V}.

The traffic flow coming from u_s (the source) is processed through the SFC (chain of the virtual nodes (VNFs)) before being delivered to u_t (the target).

For each virtual link $l_V \in E_f$, its end nodes are designated by l_V^a and l_V^b.

We propose hereafter the ILP formulation of the VNFPC problem:

1) Decision variables:

- x_u^V This variable is worth 1 if the VNF $V \in V_f$ is assigned to the node $u \in V_s$, 0 otherwise. It indicates whether the VNF V is deployed on the node u or not.
- $y_{l_s}^{l_V}$ is a binary variable which is worth 1 if the virtual link $l_V \in E_f$ is embedded on the substrate link $l_s \in E_s$.

2) Objective function:

The objective function considers minimizing the overall cost of allocating VNFs (A) and bandwidth resources (B):

$$Z = Min \quad A + B \tag{1}$$

where

- A is the overall cost of the placement of VNF instances:

$$A = \sum_{r \in R} \sum_{V \in V_f} \sum_{u \in V_s} x_u^V \times D_V^r \times \alpha_u^r \tag{2}$$

- B denotes the cost of allocating bandwidth resources:

$$B = \sum_{l_V \in E_f} \sum_{l_s \in E_s} y_{l_s}^{l_V} \times DB_{l_V} \times \beta_{l_s} \tag{3}$$

with α_u^r (resp. β_{l_s}) denoting the unitary cost of resource r (resp. bandwidth).

3) Constraints:

$$\sum_{V \in V_f} x_u^V \times D_V^r \leq C_u^r \qquad \forall u \in V_s \qquad \forall r \in R \tag{4}$$

$$\sum_{l_V \in E_f} y_{l_s}^{l_V} \times DB_{l_V} \leq CB_{l_s} \qquad \forall \quad l_s \in E_s \tag{5}$$

$$\sum_{u \in V_s} x_u^V = 1 \qquad \forall \quad V \in V_f \tag{6}$$

$$\sum_{l_s \in E_s} y_{l_s}^{l_V} \leq 1 \qquad \forall \quad l_V \in E_f \tag{7}$$

$$x_{u_s}^{B_s} = 1 \qquad x_{u_t}^{B_t} = 1 \tag{8}$$

$$x_{l_s^a}^{l_V^a} + x_{l_s^b}^{l_V^b} - y_{l_s}^{l_V} \leq 1 \qquad \forall l_V \in E_f \qquad \forall l_s \in E_s \tag{9}$$

$$x_{l_s^a}^{l_V^b} + x_{l_s^b}^{l_V^a} - y_{l_s}^{l_V} \leq 1 \qquad \forall l_V \in E_f \qquad \forall l_s \in E_s \tag{10}$$

Constraint (4) guarantees that the demand D_V^r for resources r is equal to or less than the capacity C_u^r of the substrate node u. Constraint (5) ensures that the total bandwidth allocated for any substrate link $l_s \in E_s$ must not exceed its physical link capacity CB_{l_s}. Constraint (6) ensures that a VNF $V \in V_f$ is placed in a single physical node. Constraint (7) guarantees that a virtual link $l_V \in E_f$ is placed with the use of at most one substrate link $l_s \in E_s$. Constraint (8) tells that the virtual node $B_s \in V_f$ (resp. $B_t \in V_f$) is the image of the physical source (resp. target) node u_s (rep. u_t), respectively. Equations (9) and (10) ensure that $y_{l_s}^{l_V}$ is set to 1 when the end nodes of l_V are mapped on two different substrate nodes l_s^a and l_s^b (or l_s^b and l_s^a). When the extremity nodes of l_V are mapped on the same substrate nodes, $y_{l_s}^{l_V}$ is set to 0 for all l_s to minimize the objective function.

B. Relaxed version of VNF placement and chaining problem

Here we address the relaxed version of the VNFPC problem (RVNFPC), we assume enough bandwidth on links while the resources of nodes are limited. For simplicity and without loss of generality, we will only focus on the CPU resource so we remove the $r \in R$ from the variables D_V^r and α_u^r of the general model.

For the ease of understanding, we first solve RVNFPC for negligible bandwidth costs (i.e. bandwidth is superabundant and therefore cheap), then generalize our results by considering non-negligible bandwidth costs.

For NFV-I with negligible link cost, the objective function of ILP I (Equation 1) is reduced to the A term (overall cost of the placement of VNF instances) only:

$$Z = Min \quad A = Min \sum_{u \in V_s} \sum_{V \in V_f} x_u^V \times D_V \times \alpha_u \tag{11}$$

where α_u is the unitary CPU cost on node $u \in V_s$. The following constraints stand:

$$\sum_{V \in V_f} x_u^V \times D_V \leq C_u \qquad \forall u \in V_s \tag{12}$$

$$\sum_{u \in V_s} x_u^V = 1 \qquad \forall V \quad \in V_f \tag{13}$$

Constraint (12) ensures that the demand for CPU resource D_V is equal to or less than the residual capacity C_u of each node on NFV-I. Constraint (13) guarantees that each VNF $V \in V_f$ is placed in a single node of NFV-I.

We show next that the solution to the problem described here corresponds to a variant of the Multiple Knapsack Problem (MKP) where all the objects should be put in the knapsacks. This can be proved by deriving from ILP I a new ILP II which is the same one that solves the multiple knapsack problem. This can be done by transforming the objective function (11) as follows:

$$Max \quad C(x) = Max \sum_{u \in V_s} \sum_{V \in V_f} x_u^V (\eta - D_V \times \alpha_u) \tag{14}$$

Where η is a high constant verifying:

$$\eta >> \sum_{u \in V_s} \sum_{V \in V_f} D_V \times \alpha_u$$

We note that the objective functions (11) and (14) are equivalent since:

$$\begin{aligned} \sum_{V \in V_f} \sum_{u \in V_s} \eta \times x_u^V &= \eta \times \sum_{V \in V_f} (\sum_{u \in V_s} x_u^V) \\ &= \eta \times \sum_{V \in V_f} (1) = \eta \times |V_f| \end{aligned} \tag{15}$$

Thus, minimizing objective function (14) consists to maximize non constant part of this objective (equivalent to objective function (11)), i.e,

$$Max - \sum_{u \in V_s} \sum_{V \in V_f} x_u^V \times D_V \times \alpha_u \tag{16}$$

Note that ILP II can be transformed to the multiple backpacks problem in its generic version by relaxing the constraint 13 (i.e. $\sum_{u \in V_s} x_u^V \leq 1, \forall V \in V_f$). In this case, ILP I is solvable if and only if the solution of ILP II satisfies constraint 13.

For interpretation, we say that the problem RVNFPC can be transformed to an instance of the multiple knapsack problem where the VNFs correspond to the objects and the servers correspond the knapsack. More precisely:

- The set of nodes of NFV-I is modeled by the set of knapsacks, denoted by $M = \{1, ..., m\}$.
- The SFC composed of several VNFs is modeled by the set of objects, denoted $N = \{1, ..., n\}$.
- Each VNF $V_i \in V_f$ has a demand of CPU resource denoted D_V and that is equivalent to the weight W_i of the corresponding object i in MKP.
- Placing VNF $V_i \in V_f$ on a node $u_j \in V_s$ costs $(\eta - D_V \times \alpha_u)$. This is equivalent to the profit P_i^j of placing object i in knapsack j.
- Each node $u_j \in V_s$ has a maximum CPU resource capacity denoted C_{u_j} and this is equivalent to the constant capacity of knapsack j denoted C_j in MKP.

- We naturally seek to minimize the total allocation cost of VNFs that is equivalent to maximizing the profit or negative costs of the placed objects.

For the case where the costs of links are not negligible, the objective function should be modified to include these link costs as defined by equation (3). In this way, we obtain a new variant of the MKP where the cost of any object i depends on the knapsacks selected to bring the object i, its precedent object $i-1$ and its next object $i+1$.

When the costs of the paths interconnecting the VNF servers are equal to or much higher than the CPU costs, the optimal placement and chaining of an SFC with equal bandwidth demands is obtained by solving the NP-hard bin backing problem if all the CPU costs are identical.

IV. GENETIC ALGORITHMS BASED META-HEURISTIC

To solve the RVNFPC problem which is NP-hard, we propose to use genetic algorithms (GA) which are efficient in solving almost all variants of the knapsack problem. GA allows us to find the near optimal solution among a set of feasible solutions thanks to a fitness function. The GA works as follows.

1) Initial population: as start with an initial population that is used to diversify and generate new solutions by applying crossover and mutation operations. The selection of an initial population with individuals of different and varied genes is crucial to explore a maximum of promising regions in the solutions space while the inclusion of individuals with higher quality genes in the initial population generally leads to rapid convergence towards the best solutions. In our proposal, the initial population is generated according to the following processes:

- Random Generation: for each SFC of n VNFs, a list of n servers randomly selected is generated and evaluated according to the fitness function.
- Constrained Shortest Paths: For an SFC composed of n VNFs, a shortest path of n links connecting the SFC's source to a neighbor of the SFC's destination is determined. The constraints are checked at the path calculation stage so that only paths satisfying the constraints are determined and added to the initial population. Zero-cost reflexive links with unlimited capacities are added to NFV-I to allow deployment of multiple successive VNFs on a same server. In order to get enough individuals in the initial population, random solutions generated with the precedent process are added to the initial population.

2) Coding: Coding is a function that transforms real data of the problem to data used by the GAs. In our approach, we used a coding with natural numbers representing the indices of servers. We genetically code a VNF placement operation

1	2	3	4	5	6	7	8	9	10
3	4	9	2	1	4	10	6	8	3

Fig. 3: Chromosome coding example

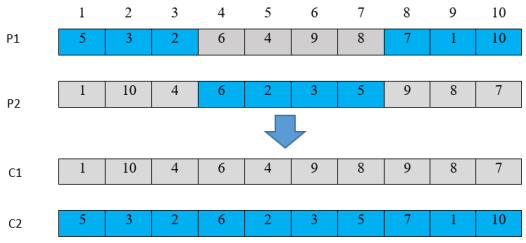


Fig. 4: Crossover example

as follows: each VNF placement solution corresponds to an individual that is coded by an array where the indices correspond to the VNF identifiers and the cell contents to the servers on which the VNFs are deployed. For an SFC with 10 VNFs and a NFV-I with 10 servers, an example of solution coding is shown in Figure 3 where *VNF1* and *VNF10* are placed on *Server 3* whereas *VNF3* is placed on *Server 9*.

3) Fitness function: fitness measures the quality of individuals and controls the process of breeding: the higher the fitness of an individual, the greater the likelihood of using that individual for breeding. In our GAs, fitness is determined as follows:

- For feasible solutions:

$$fitness = cost^{-(1+\frac{index}{nb\_generations})}$$

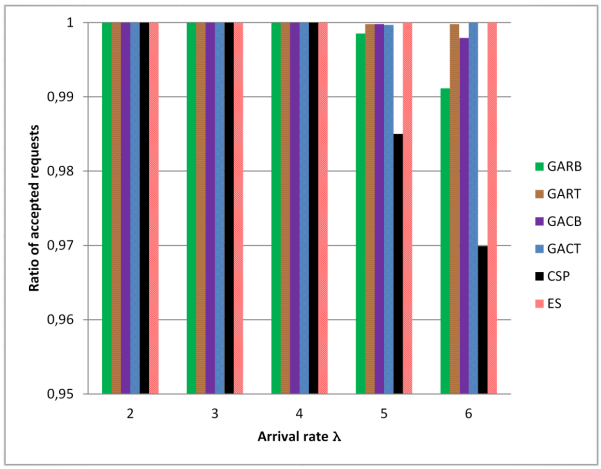
  where $index$ corresponds to the number of times the current population is regenerated (i.e. $index$ varies from 1 to $nb\_generations$), $nb\_generations$ corresponds to the total number the population is regenerated and $cost = A + B$ is determined according to equations (2) and (3). In this way, the probability that an individual's genes are derived from the best individuals (i.e., individuals with smallest costs) increases over time.
- For non-feasible solutions: in order to prevent the algorithm from stopping quickly without exploring the promising regions in the solution space (specifically when the initial population does not contain any feasible solution), we associate very small fastnesses to non-feasible solutions. For breeding, non-feasible solutions that violate fewer constraints should be preferred. In our proposal, we chose to run through the individual's genes in the same order and count the number of times ($nb\_violate$) constraints are violated:

$$fitness = \frac{\epsilon}{nb\_violate}$$

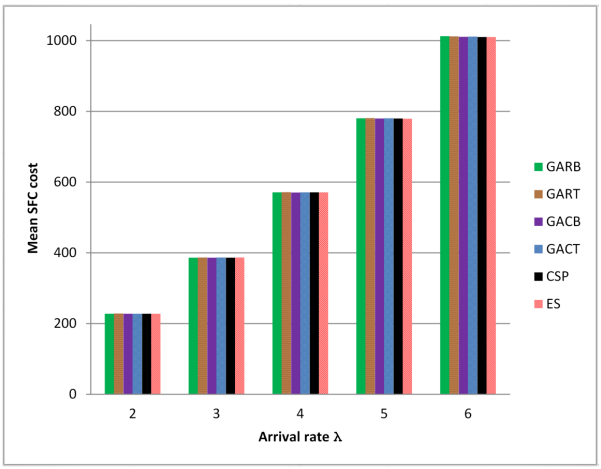
  where ϵ is a very small constants.

4) Genetic operators: Population reproduction is the process of generating a new population $i+1$ from a previous population i. This process is constituted by the use of the selection, crossover and mutation operations:

- Selection: this process helps to determine which individuals are more inclined to obtain the best results. For our algorithms, we used two selection processes: (1) tournament selection where parents and children are



(a) Ratio of acceptation



(b) SFC cost

Fig. 5: Comparison results for various arrival rates and small NFV-I and SFC sizes

randomly matched before the best individuals in pairs are selected, and (2) selection of the best individuals where the next population is composed of individuals with the highest fitness.

- Crossover: crossover is a reproduction operation which allows the exchange of genetic information between individuals. It uses two parents to produce one or two children. In our proposal, the parents are selected according to the *Roulette Wheel Selection* where the probability of choosing an individual for breeding of the next generation is proportional to its fitness.
  For children generation, we used the 2-points crossover where the points are randomly selected among the genes. In the example of Figure 4, the genes between the two cross point indices $(4, 8)$ in individuals P1 and P2 are colored in light gray and blue respectively while the other genes are colored in blue and light gray respectively. By swapping the genes of *P1* and *P2* located between the crossing indices, we obtain two new children *C1* and *C2* as shown in Figure 4.
- Mutation: The role of this process is to randomly modify, with a certain probability, the value of a component of the individual. In our solution, we randomly choose a gene then we randomly replace the identifier of the server S with another server identifier S' drawn randomly from the list of servers.

V. Performance Evaluation

In this section, we first present the compared algorithms and describe the simulation environment, then, we define the performance metrics to evaluate our proposal and finally present the simulation results.

A. Compared algorithms

The compared algorithms in our simulation are:

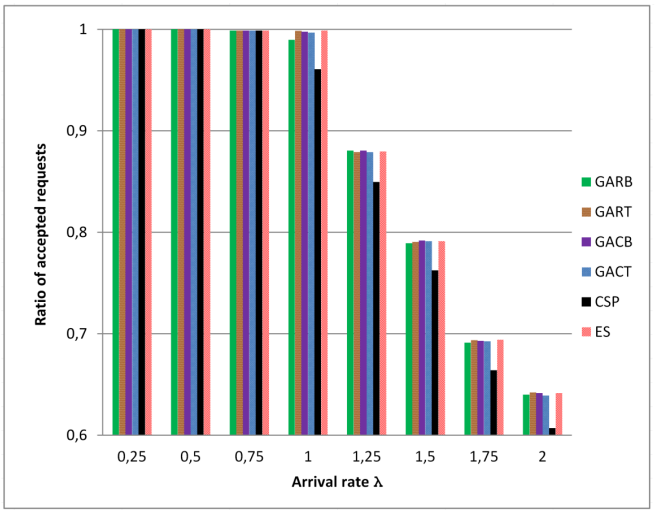
- GA-**RB**: is a genetic algorithm with **R**andom generation of the initial population and selection of the next generation by choosing the **B**est individuals among children and parents.
- GA-**RT**: is a genetic algorithm with **R**andom generation of the initial population and selection of the next generation by making a 2 to 2 **T**ournament between an individual belonging to the different parents and an individual belonging to the children. The parents are paired with the children in a completely random way.
- GA-**CB**: identical to GA-RB but the initial population consists of Shortest **C**onstrained Paths to the destinations' neighbors. These paths will be complemented by other randomly generated mapping solutions to gather a population containing a sufficient number of individuals.
- GA-**CT**: identical to GA-RT but the initial population consists of Shortest **C**onstrained Paths to the destinations' neighbors.
- CSP: **C**onstrained **S**hortest **P**aths between the sources and destinations in terms of cost. The constraints are checked throughout the calculation to ensure that the determined paths satisfy them. A constraint vector is associated with each node and a modified version of the Ford-Bellman algorithm is run to determine the best paths verifying the constraints.
- ES: **E**xhaustive **S**earch to determine the solution that minimizes the cost.

The population size is set to 100 individuals and the mutation probability is 0.01. Simulation time is 10^5 units.

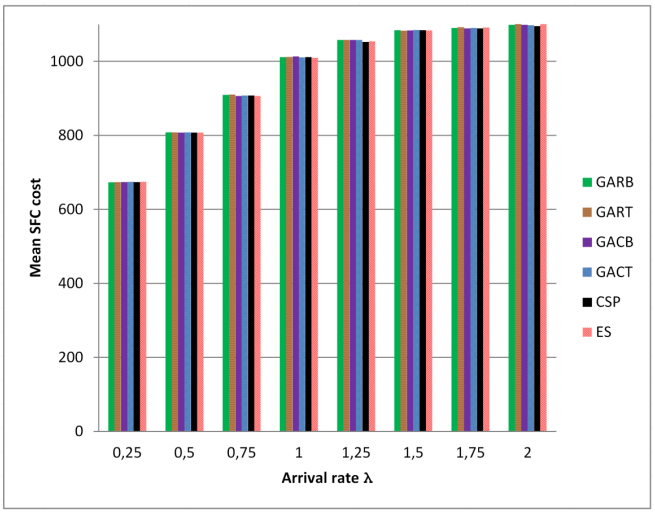
B. Simulation Environment

SFC requests arrive according to a Poisson process with λ requests/time unit and the lifetime of each request follows an uniform distribution $U(10, 20)$.

We evaluated our algorithms on two NFV-Infrastructures, the first one is composed of 30 nodes and the second one is composed of 100 nodes. The available CPU capacity per physical node is randomly drawn within the range of 20 to 120 units and the available bandwidth capacity per link is



(a) Ratio of acceptation



(b) SFC cost

Fig. 6: Comparison results for different network loads and medium NFV-I size

1000 units. The requested CPU capacity of each VNF is drawn randomly in (10, 20) with random requested bandwidths in the (2, 5) interval. The resources costs α and β are randomly generated in the range of (5, 20).

C. Performance Metrics

In this subsection, we define the simulation metrics used for the performance evaluation and comparison purposes.

1) Mean SFC Cost (MSC): this metric MSC determines the average cost of SFCs that are successfully placed. It corresponds to the ratio between the total cost of CPU and bandwidth resources allocated for SFCs and the total number of accepted SFCs.

2) Ratio of Accepted requests (RA): we define RA as the ratio between the number of accepted SFC requests and the total number of received SFC requests.

D. Simulation Results

In our simulation, we first compared our proposals to ES and CSP for small sizes of SFCs and NFV-I, then compare our variants of GAs with CSP for medium and high sizes of SFCs and NFV-I.

In our first experiment, we compared our 4 variants of GAs with ES and CSP for SFCs of n VNFs ($n \in (2, 6)$) and an NFV-I with 30 nodes. The results depicted in Figure 5 clearly shows that the various compared algorithms have very close performance. Indeed, although RA of CSP seems slightly lower than those of GAs and ES for SFCs with 6 or 5 VNFs (see Figure 5a), the differences are negligible and indistinguishable for the other scenarios.

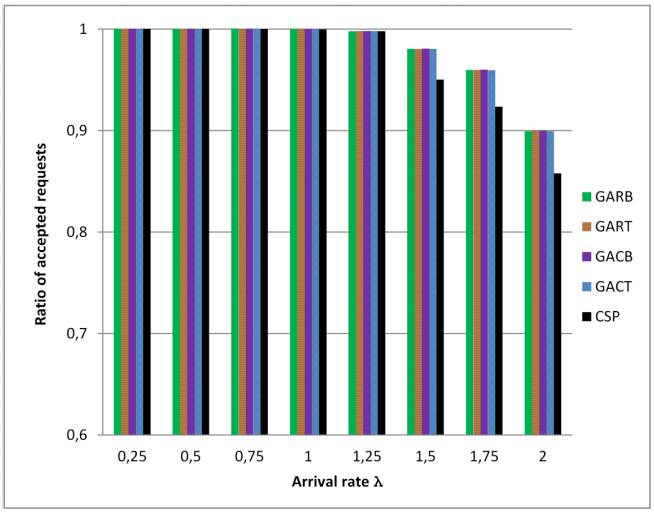
Whereas ES guarantees the determination of existing optimal solutions, the probability of CSP to determine solutions decreases with the increase of SFC sizes. In fact, the more the sizes of the SFCs increase, the more the lengths of paths increase. Since CSP only maintains one path per node during the computations, the probability of determining the best solutions decreases with increasing SFC sizes (as the number of paths of $x + 1$ links is greater than the number of paths of x links). For example, for SFCs of 2 VNFs, CSP returns optimal solutions.

To complete our first comparative study, we submitted the compared algorithms to various network loads. In our second experiment, we used an NFV-I of 30 nodes and generate randomly SFCs of 6 VNFs.

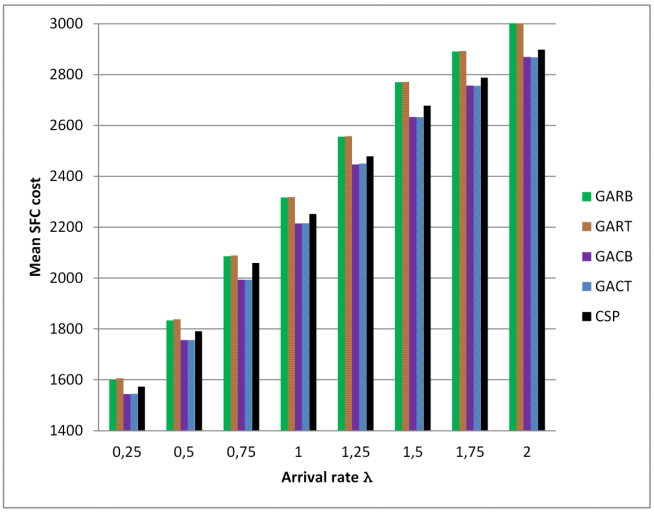
The results depicted in Figure 6 show that our variants of GAs have performance very close to ES that is slightly better than that of CSP for RA. Figure 6a shows that more the network load increases, less the ratio of accepted request is higher. At higher loads, CSP has lower but close performance than those of ES and GAs. This can be explained by the operation of CSP which does not explore all the promising regions in the solution space since it does only maintain one best path per node. Besides, CSP tends to prefer reusing the same path segments (since it explores nodes in the same order), which could lead to the overloading of some parts of the network and, thus reduces the probability of determining solutions for the future requests.

Concerning MSC, Figure 6b shows that the various compared algorithms are very close although the determined solutions are sometimes different. Like CSP that only maintains one best path segment per node to built the best solution, GAs often construct their solutions by combining the path segments of the best solutions. Thus, both these algorithms determine nearly optimal solutions although CSP is not able to explore some promising parts of the solution space.

Our two first experiments with small SFCs and NFV-I sizes clearly show that GAs and ES have similar performance which are slightly better than those of CSP, especially for RA. For a more complete comparison study and in order to better understand the behavior of our variants of genetic algorithms, we extended the problem size by increasing the size of SFCs



(a) Ratio of acceptation



(b) SFC cost

Fig. 7: Comparison results for different network loads and large NFV-I size

(size in (10, 20)) and NFV-I (100 nodes).

Figure 7 shows the results where we see that GA variants GACB and GACT outperforms the other algorithms CSP, GARB and GART.

With regard to RA (Figure 7a), all the GA variants are better than CSP. This means that diversification and increase of the number of path segments used for the computation can slightly improve RA.

Concerning MSC, Figure 7b shows that the GA variants (GACB and GACT), which include random and constrained shortest paths in the initial population, allow to decrease the mean SFC cost compared to the other algorithms.

With the variants starting with random paths, the SFC costs are slightly larger than those of CSP. This can be explained by the small population size and the low running time we devoted to the computations on the one hand, and to the rejection of the largest SFCs with CSP in the other hand (recall that RA of CSP is lower that those of GAs).

If the choice of the initial population seems crucial to improve the performance of GAs, Figure 7 show that the selection process of the next population has negligible impact on the global performance. Besides, the choice of adding constrained shortest paths to the starting population could accelerate the convergence and improve the performance.

VI. Conclusion

This paper addresses the VNF placement and chaining problem for NFV environment. After formulating the problem with ILP, we relaxed the link constraints and transformed the problem to the well known knapsack problem. In this way, we proposed efficient genetic algorithms based meta-heuristic to solve large instances of the problem.

Our proposal is evaluated through extensive simulations. Performance comparisons with relevant algorithms demonstrate the effectiveness of our algorithm in optimizing allocation costs and increasing the ratio of accepted requests.

References

[1] Kaur, K.; Mangat, V. and Kumar, K. A comprehensive survey of service function chain provisioning approaches in SDN and NFV architecture Computer Science Review, 2020, 38, 100298

[2] Kim, S. I. and Kim, H. S. A VNF Placement Method based on VNF Characteristics, 2021 International Conference on Information Networking (ICOIN), 2021, pp. 864-869, doi: 10.1109/ICOIN50884.2021.9334022.

[3] Yi, B.; Wang, X.; Li, K.; k. Das, S. and Huang, M. A comprehensive survey of Network Function Virtualization Computer Networks, 2018, 133, 212 - 262

[4] Li, D.; Hong, P.; Xue, K. and Pei, J. Virtual network function placement and resource optimization in NFV and edge computing enabled networks Computer Networks, 2019, 152, 12 - 24

[5] Gadre, A.; Anbiah, A. and Sivalingam, K. M. Centralized approaches for virtual network function placement in SDN-enabled networks EURASIP Journal on Wireless Communications and Networking, 2018, 2018, 197.

[6] Gupta, A.; Farhan Habib, M.; Mandal, U.; Chowdhury, P.; Tornatore, M. and Mukherjee, B. On service-chaining strategies using Virtual Network Functions in operator networks Computer Networks, 2018, 133, 1 - 16

[7] Soualah, O.; Mechtri, M.; Ghribi, C. and Zeghlache, D. Online and batch algorithms for VNFs placement and chaining Computer Networks, 2019, 158, 98 - 113.

[8] Wahab, O. A.; Kara, N.; Edstrom, C. and Lemieux, Y. MAPLE: A Machine Learning Approach for Efficient Placement and Adjustment of Virtual Network Functions Journal of Network and Computer Applications, 2019, 142, 37 - 50.

[9] Ghaznavi, M.; Shahriar, N.; Ahmed, R. and Boutaba, R. Service Function Chaining Simplified CoRR, 2016, abs/1601.00751.

[10] Yi, B.; Wang, X. and Huang, M. Design and evaluation of schemes for provisioning service function chain with function scalability Journal of Network and Computer Applications, 2017, 93, 197 - 214.

[11] Soualah, O.; Mechtri, M.; Ghribi, C. and Zeghlache, D. An efficient algorithm for virtual network function placement and chaining 2017, 647-652

[12] Mechtri, M.; Ghribi, C. and Zeghlache, D. A Scalable Algorithm for the Placement of Service Function Chains IEEE Transactions on Network and Service Management, 2016, 13, 1-1

[13] Morin, C.; Texier, G.; Caillouet, C.; Desmangles, G. and Phan, C.-T. VNF placement algorithms to address the mono- and multi-tenant issues in edge and core networks CLOUDNET 2019 : 8th IEEE International Conference on Cloud Networking, 2019

[14] Khebbache, S.; Hadji, M. and Zeghlache, D. Virtualized network functions chaining and routing algorithms Computer Networks, 2017, 114, 95 - 110.