A scalable algorithm for the placement of service function chains

Marouen Mechtri, Chaima Ghribi, Djamal Zeghlache

A Scalable Algorithm for the Placement of Service Function Chains

Marouen Mechtri, Chaima Ghribi, and Djamal Zeghlache
Télécom SudParis, Samovar-UMR 5157 CNRS, Université Paris-Saclay

***Abstract*—Network Function Virtualization (NFV) decouples software implementations of network functions from their hosts (or hardware). NFV exposes a new set of entities, the virtualized network functions (VNFs). The VNFs can be chained with other VNFs and physical network functions (PNFs) to realize network services. This flexibility introduced by NFV allows service providers to respond in an agile manner to variable service demands and changing business goals. In this context, the efficient establishment of service chains and their placement becomes essential to reduce capital and operational expenses and gain in service agility. This paper addresses the placement aspect of these service chains by finding the best locations and hosts for the VNFs and to steer traffic across these functions while respecting user requirements and maximizing provider revenue. We propose a novel eigendecomposition based approach for the placement of virtual and physical network functions chains in networks and cloud environments. A heuristic based on a custom greedy algorithm is also presented to compare performance and assess the capability of the eigendecomposition approach. The performance of both algorithms is compared to a multi-stage based method from the state of the art that also addresses the chaining of network services. Performance evaluation results show that our matrix based method, eigendecomposition of adjacency matrices, has reduced complexity and convergence times that essentially depend only on the physical graph sizes. Our proposal also outperforms the related work in provider's revenue and acceptance rate.**

***Index Terms*—Virtual Network Function, Function placement and chaining, Eigendecomposition, Distributed Cloud environments.**

I. Introduction

NETWORK Functions Virtualization (NFV) is revolutionizing the way networking services are designed and deployed. Compared to traditional networking where dedicated hardware is required for each function (manually installed into the network), the NFV concept virtualizes the network functions (NAT, firewalling, intrusion detection, DNS, caching...) so they can be hosted on commodity hardware (servers/computers). These functions decoupled from the underlying hardware are known as Virtualized Network Functions (VNFs).

Running NFV based networks provides considerably more flexibility, leads to efficient and scalable resource usage and reduces costs. Operators (providers) are increasingly interested in Virtual Network Functions (VNFs) that can be placed in Data Centers or NFV-capable network elements such as routers and switches. Despite the emergence of NFV, deploying and orchestrating VNFs still requires more research and development. The challenge of VNF and VNF forwarding graph (VNF-FG) placement in the cloud requires more attention. The notion of VNF forwarding graphs is defined by ETSI as service chaining that consists in steering traffic flows across switches and VNFs in an ordered fashion. A service chain is a topology of services, or more specifically a sequence of VNFs interconnected to provide a complex network service with specific functionalities. Figure 1 depicts the notion of VNF chaining that consists in setting up a service chain (represented as a VNF forwarding graph (VNF-FG)) on virtual and physical provider service nodes.

The focus of this paper is the problem of virtual network function (VNF) placement and chaining across distributed cloud environments. Previous work on VNF chaining and placement typically maps the problem into a traditional Virtual Network Embedding (VNE) problem [1], [2], [3]. Even if VNF placement and routing seem to be similar to the well known VNE problem, they are are different [4]. Requests in VNE are modeled by simple undirected graphs (multipoint-to-multipoint connections) whereas in VNF chaining, demands are more complex and contain both the VNFs to place and the traffic flows to steer between source-destination pairs (point-to-point/ingress-egress connections).

Previous work in [5], [6], [7], [3] and [8] solves the problem of VNF placement and flow steering separately. Two-stage placement and chaining approaches cannot provide efficient solutions especially for large input graphs which are hard to map. In [9], authors propose a Mixed Integer Linear Programming (MILP) formulation to the problem of joint optimization of VNF placement and workload distribution and a greedy solution to solve the problem for every VNF chain one by one. Authors in [6], [7], [3] propose exact and heuristic algorithms. Despite their optimality, the exact solutions suffer from combinatorial explosion and do not scale with problem size. Heuristic approaches converge faster but solve the problem iteratively and this affects the quality of the solutions and increases the time to find a suboptimal solution.

In this paper, we formalize the VNF placement and chaining problem across distributed clouds and propose an analytical approach to provide a more efficient solution. This approach is based on the eigendecomposition of the request and infrastructure graphs that we extend to make it applicable to the VNF placement and chaining problem. To assess the performance

M. Mechtri and C. Ghribi are with the Department of Wireless Networks and Multimedia Services, Institut Mines-Telecom, Telecom SudParis, Evry, France, e-mails: {marouen.mechtri, chaima.ghribi}@it-sudparis.eu.

D. Zeghlache is the head of Wireless Networks and Multimedia Services Department, Telecom SudParis, IMT and member of Université Paris-Saclay e-mail: djamal.zeghlache@it-sudparis.eu.

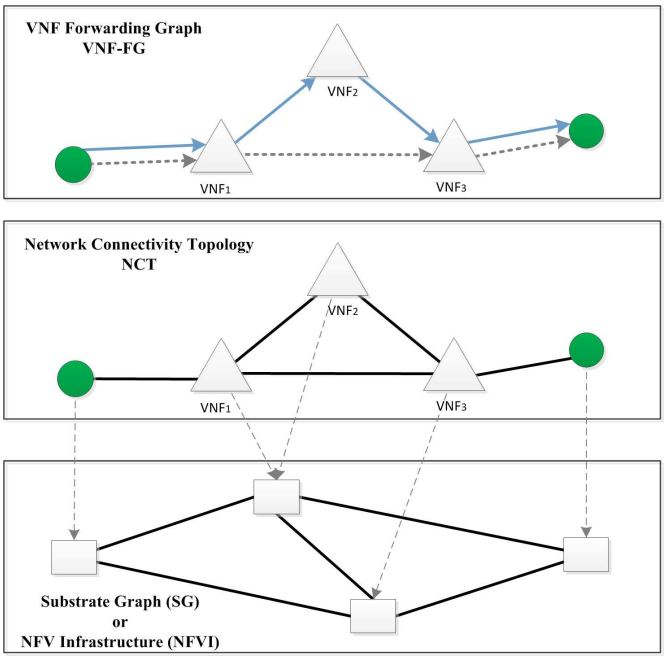

Fig. 1: VNF placement and chaining

of our solution we compare with a heuristic greedy procedure also proposed to solve the problem iteratively. Both approaches are evaluated considering several metrics and use cases such as infrastructure size, request size, network connectivity and system load.

Section II of this paper presents the related work and highlights the contribution of our customized eigendecomposition approach to VNF placement and chaining. Section IV formulates the problem and Section V describes the proposed Eigendecomposition method. Our greedy heuristic algorithm is discussed in detail in Section VI. Performance evaluation results and comparison with a selected state of the art algorithm are reported in Section VII.

II. Related Work

There has been much recent work in the area of VNF chaining and placement. A preliminary work was proposed in [7], authors formulated the VNF placement problem as a integer linear program (ILP) and proposed a simulated Annealing (SA) based heuristic to get approximate solutions in shorter time but have simplified the overall problem such as using only one type of VNFs and addressing rather small chains.

Authors in [10] introduced ClickOS, a platform to manage software-based middle-boxes. In [5], authors presented Stratos, a network-aware orchestration layer for virtual middleboxes in Clouds and expressed middlebox provisioning as an ILP problem.

In [11], authors propose a NFV based orchestration framework for enterprise WLAN. The orchestrator implements a recursive greedy algorithm based on breadth-first traversal search which aims at placing VNFs according to application level constraint.

In [6], authors formulate the problem of network function placement and routing as a mixed integer linear programming problem to determine the placement of services and the routing of the flows while minimizing resource utilization. Heuristic solutions were also proposed to solve the problem incrementally.

In [12], authors propose an Integer Linear Programming (ILP) model to solve the network function placement and chaining problem. Additionally, they propose a heuristic procedure that employs a modified version of the proposed ILP model in each iteration.

A model for formalizing the chaining of network functions using a context-free language is proposed in [2]. Authors describe the mapping as a Mixed Integer Quadratically Constrained Program (MIQCP) for finding the placement of the network functions and chaining them together.

In [13], authors describe an architecture based on an orchestrator that ensures the automatic placement of the virtual nodes and the allocation of network services on them. Placement decisions are made using simple algorithms, such as counting the number of virtual routers on a host, or using algorithms based on a set of constraints and policies that represent the network properties.

A model for resource allocation in NFV networks was proposed in [3]. The model can be used in both pure NFV networks and in hybrid networks containing physical hardware. The presented model was implemented as an ILP and was evaluated using a service provider scenario containing two types of service chain requests.

A network functions virtualization orchestration model was provided in [14]. Authors define and formulate the VNF Placement and Routing optimization problem via mathematical programming and design a greedy math-heuristic algorithm to solve it. A service chaining algorithm was also proposed in [15]. Authors formulate a Genetic Programming based approach to solve the VM allocation and network management problem.

Existing research work is generally based on mixed integer programming to solve the VNF placement and chaining problem. Even if these approaches allow finding exact solutions, they suffer from combinatorial complexity, do not scale with problem size and are extremely time-consuming. The existing proposed greedy heuristic approaches provide solutions in a more reasonable time but they solve the problem iteratively which affects the quality of the solutions and wastes time. Demands are almost formulated as single service chains expressing a sequence of VNFs that should be applied to a flow (e.g, [16] [7]). In some works VNF chaining requests are represented as connected graphs (e.g, [15] [12]) but place service chains heuristically, one by one, in an iterative way that leads to suboptimal solutions. We aim at placing chains simultaneously on the hosting infrastructures.

Authors of paper [17] propose an ILP and a heuristic approach for VNF placement and chaining using a complex transformation of the problem by adding new virtual switches. Their heuristic first models the VNF orchestration problem as a Multi-Stage directed graph with associated costs. Second they achieve VNF placement by running the Viterbi algorithm [18]

on the Multi-Stage graph. The Multi-Stage algorithm favors mapping nodes of each request on the same physical node. This means that bandwidths on the links are neglected since there are no link constraints when VNFs are mapped on the same physical node. Link constraints and requirements are naturally embedded in our models and solutions.

Authors of [19] treat the problem of network service chaining through an ILP model and a heuristic-based algorithm composed of two phases: a decomposition selection with backtracking phase and a mapping phase, leading consequently to suboptimal solutions.

In most of the above mentioned solutions, the VNF placement and chaining problem is solved in two steps: VNFs are initially mapped to substrate nodes then the traffic is steered through chains via the shortest possible path. As opposed to two-stage approaches, our solution solves the problem jointly. VNFs are placed and traffic is distributed over them in one-shot since tenant requests are collectively processed as VNF Forwarding Graphs (Service Chains).

In summary, our work is different from the existing VNF placement and chaining proposals since it relies on an a new analytic Eigendecomposition approach that handles joint VNF placement and traffic distribution. Our solution is also much faster and more scalable compared to existing algorithms and it naturally achieves tuned consolidation for better resource usage. We have also supported the concept of multi-tenancy and considered both virtual and physical resources. To the best of our knowledge, this work is the first effort that uses Eigendecomposition to solve the problem while improving resource usage, fast converging to solutions and scaling with problem size.

III. VNF CHAINING AND USE CASES

The chaining of network functions was investigated by different working groups and research projects in order to define the issues associated with the deployment of service functions. The ETSI-NFV and the SFC IETF working groups specify the NFV architecture [20] along with its components and the associated terminologies [21].

In Figure 1, stemming from ETSI-NFV, we distinguish three types of graphs that represent the requested network service. A virtualized network function forwarding graph (VNF-FG) that represents the consumer or tenant request, a network connectivity topology (NCT) that corresponds to the underlying network necessary to support the forwarding graph flows, and the substrate graph or the NFV infrastructure (NFVI) that will host the requested resources. In the ETSI-MANO document [22], the requested graphs are defined as follows:

- Network connectivity topology (NCT): a graph that specifies the VNF nodes that compose the global network service and the connection between these nodes through virtual links (VL). Each VL is connected to a VNF through a connection point (CP) that represents the VNF interface.
- VNF Forwarding Graph (VNF-FG): is a graph established on top of the NCT. The VNF-FG is composed of network forwarding paths (NFP) that are ordered lists of CPs that form a VNF chain.

Figure 2 shows an example of a network service composed of:

- End points that represent the ingress and egress nodes of the chain. These nodes can be physical nodes (e.g. switches) or virtual nodes (e.g. VMs hosting a web server application) from which the chain and the offered service are reachable. Packets from the end points will traverse a specific chain based on their traffic types.
- Classifiers that identify and classify traffic, based on policy specified by the tenant/operator, into service flows to be shepherd to the appropriate forwarding path. Classifiers impose the classification and implement the policy by applying the encapsulation method on the packets (e.g. via the Service chaining protocol: Network Service Header protocol "NSH" [23]). This technique ensures packet delivery to the requisite VNFs. SFC working group of IETF recommend the use of the classifier in the service chaining domains. Note that the classifier can also be co-located with a VNF.
- NFPs representing the network connectivity between a set of VNFs forming the requested chain (e.g. a chain of VNFs such as for example firewalls, NATs, and load balancers on the path to a web server). Figure 2 depicts an example of two forwarding paths "NFP1 and NFP2". NFP1 traverses VNF1, VNF2 and VNF3 through their associated CPs whereas NFP2 traverses only VNF1 and VNF3.
- VNFs which are virtual containers implementing a specific network function. In our model we consider also physical network functions (PNFs).

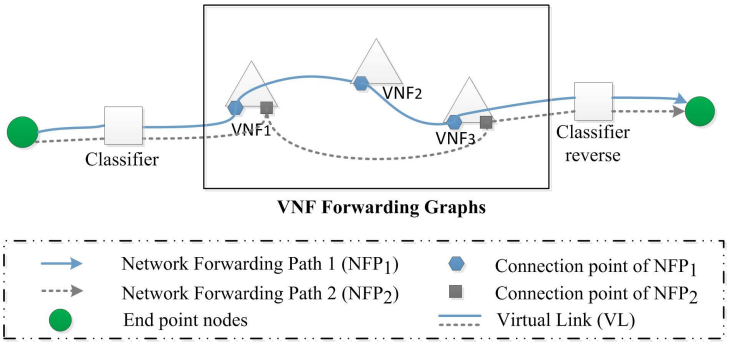

Fig. 2: NFV components and terminologies

The ETSI-NFV and IETF-SFC working groups propose their own terminologies to express respectively the main concepts of NFV and SFC. The terminologies as usual are different but their is a certain correspondence between their components and concepts as presented in Table I.

ETSI-NFV		IETF	
Virtualized network function	VNF	Service function	SF
Connection point	CP	Service Function Forwarder	SFF
Virtual link	VL	implicitly	–
VNF forwarding graph	VNF-FG	Service Function Chain	SFC
Network Forwarding path	NFP	Service Function Path	SFP

TABLE I: ETSI-NFV and IETF Notations

Before we delve into the analysis of the use cases, our approach and model, the reader should be aware that the ETSI-NFV and IETF-SFC recommendations for the implementation of the network service chains requires the use of connection points and a classifier that sets (establishes) and imposes the direction of the chain. Our modelling is based on this principle and goes through the optimization of the NCT placement followed by its instantiation via virtual infrastructure managers (VIMs) of the NFVI. For the optimization we will use the derived NCT as input. The implementation of the optimized chains is achieved by a NFV manager that instantiates and stitches the connection points and the associated forwarding paths. Our choice is governed by the fact that in cloud environments upstream and downstream flows are tightly coupled on a physical link. If such a constraint disappears in the future, we would simply use the initial VNF-FG as the requested (input) graph in the optimization. Note that we assume that the tenant/client specification embeds any special demand including VNF replication and link splitting in their requested graphs (VNF-FG). Hence, our main goal is to solve the optimization problem in a realistic, thus constrained, environment.

A. *Use Cases Overview*

The IETF-SFC and ETSI-NFV [24] working groups present in several documents network function chaining use-cases in, for instance, mobile networks [25] and data center networks [26]. In Cloud and data center environments, we distinguish two primary types of traffic/communication between cloud users and services. The first type is a North-south traffic that corresponds to a communication between an external entity and a service hosted in the Cloud or the data-center (e.g. VPN service). An example that illustrates this scenario is a cloud provider that deploys the appropriate network functions (on dedicated network hardware or on virtual machines) and the associated forwarding path in order to offer and expose its service. The second type is an East-West traffic where the communication occurs between resources deployed inside the Cloud environment. For this scenario, the network service chain is deployed on the path between the tenant's applications (e.g. between a web server and a database).

We summarize the use cases from ETSI-NFV and IETF-SFC for which the VNF-FGs and network service chains can be addressed by our approach and our proposed eigende-composition method. This analysis of the use cases helps us extract common (generic)patterns and relevant chain structures to address using our algorithms while achieving the largest possible coverage of the network services chaining and placement problem.

- **Use case 1: VNF-FG deployment for in data center traffic:** Tenants can request complex services with compute resources connected via VNF chains when the traffic originates from inside the data center. As shown in the figure below, the VNF-FG is composed of Chains with several virtual machines (VMs) and VNFs.
- **Use case 2: VNF-FG deployment for intra and inter data center traffic:** In this use case (north-south use case defined in [26]), the traffic is staying within the data center but is originating from a remote data center or a user through an edge router connecting to an external network. This can refer to VNF chains spanning multiple data centers, thus requiring inter-data-center coordination [27]. In this use case, the focus is on hybrid chains composed of both physical and virtual network functions. In this scenario, NFP_1 and NFP_2 traverse the same physical network function PNF_1. This scenario describes the case of a cloud (or data center) provider imposing the traversal of a physical network function before reaching the desired service (e.g. a physical firewall).

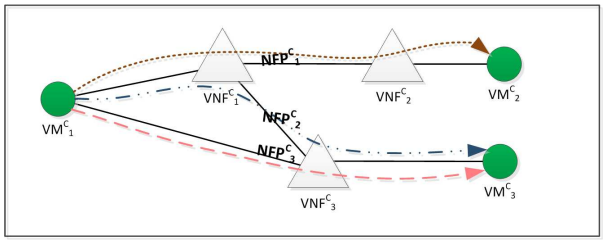

Fig. 3: VNF-FG deployment for in data center traffic

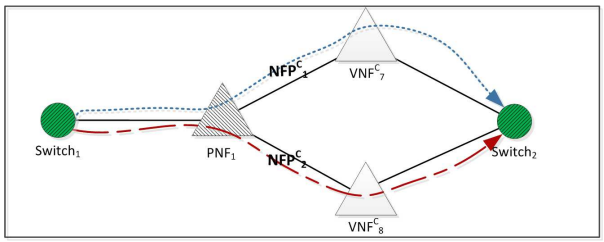

Fig. 4: VNF-FG deployment when traffic originates from outside the data center

The problem that we aim to resolve should not only support the cited scenarios and use cases but also be sufficiently generic to apply to infrastructures and environments with virtual and physical network functions.

IV. Problem Formulation

The VNF placement and chaining problem in the Cloud corresponds to mapping the tenants VNF forwarding graphs into provider's infrastructures (or NFVI) and can be decomposed into two subproblems: Placement and Chaining. "Placement" consists in selecting the substrate/service nodes (server or switch) that will host the requested VNFs while "Chaining" consists in creating paths or steering traffic through VNFs.

A. *Substrate graph or NFV Infrastructure*

The infrastructure (or substrate) is modeled as a weighted undirected graph $SG = (N^s, E^s)$ where N^s represents the set of substrate nodes and E^s the set of substrate links. Each substrate node $n_i^s \in N^s$ has a physical-host-type t ($t \in \{server, switch, PNF\}$) and is associated with an available processing capacity $cpu(n_i^s)$. Each substrate link $e^s(i,j) \in E^s$ connecting nodes n_i^s and n_j^s is associated with

an bandwidth capacity $bw(e^s(i,j))$. Each node of the substrate graph has a specific physical-host-type such as a server, a switch or a PNF (physical network function) node. On the server node, we can host a virtual machine acting as a VNF. On the PNF node we can host a VNF having the same VNF-type as the PNF so we can impose for a VNF to be hosted on a PNF with a specific type. Consequently the VNF-type feature can be used to make sure mapping occurs on similar node types or to enforce mapping on a node with specific features.

B. *Virtual Network Function forwarding graph*

Requests are modeled as virtual network function forwarding graphs. Let $VNF\text{-}FG^c = (N^c, E^c)$ be the VNF forwarding graph associated to a tenant c. The set of nodes N^c represents the set of requested services. A requested service node $n_i^c \in N^c$ can represent an end-point node (a VM or a switch) and is associated to a requested processing capacity $cpu(n_i^c)$ according to its type. E^c is the set of links of the VNF forwarding graph.

Let $NFP^c = \{NFP_1^c, ..., NFP_{K^c}^c\}$ be the set of Network Forwarding Paths (or chains) requested by the tenant c. Each Forwarding Path NFP_j^c describes the ordered VNF sequence the traffic must pass through and is associated to a requested bandwidth capacity $bw(NFP_j^c)$.

From the VNF forwarding graph $VNF\text{-}FG^c$, we derive a request graph called network connectivity topology graph NCT^c. $NCT^c = (N^c, E^c)$ is a directed weighted graph having exactly the same set of nodes and edges of $VNF\text{-}FG^c$. The weight of a node $n_i^c \in N^c$ corresponds to its requested processing capacity $cpu(n_i^c)$. The weight (or requested bandwidth) of an edge $e^c(i,j) \in E^c$ corresponds to the sum of the bandwidths of all the VNF flows passing through it.

As specified by the SFC IETF working groups [26], we associate a VNF-type to each VNF to represent the network service or function type (e.g., firewall, DPI, NAT, etc.). By default, VNF nodes will be hosted on physical servers, but a tenant can impose mapping a VNF on a PNF. We rely on the VNF-type matching to ensure this required mapping. For example, a tenant can request to map a VNF on a server supporting SR-IOV or acceleration technology. This can be done if we define a PNF with an SR-IOV type or a VMDq-type for mapping on a node with VMDq technology.

C. *Objectives*

Our main objective is to ensure joint VNF placement and chaining when targeting the most complex and complete use cases proposed by ETSI [24] (dealing with VNF-FGs). Many important aspects and concepts like multi-tenancy and resource heterogeneity (Virtual/Physical) should be also considered:

- Supporting multi-tenant aware service models is strongly required by IETF specification [26]. Multi-tenant service delivery is achieved when providers' substrate nodes (SNs) are tenant aware (SNs can be allocated to multiple tenants) or when SN instances are tenant dedicated.
- Service chaining is generally limited to exclusively physical or virtual SNs and not a mix of resources [26]. Considering heterogenous resources presents the advantage of combining the benefits offered by physical SNs (performance) with the flexibility and agility provided by virtual SNs.

Symbol	Description
$SG = (N^s, E^s)$	SG is a substrate network/graph with a set of nodes N^s and a set of links E^s.
n_i^s	Substrate node j. $n_j^s \in N^s$.
$cpu(n_i^s)$	Available processing capacity of n_j^s.
$e^s(i,j)$	Substrate link $\in E^s$ connecting nodes n_i^s and n_j^s.
$bw(e^s(i,j))$	Available bandwidth capacity of $e^s(i,j)$.
$VNF-FG^c = (N^c, E^c)$	VNF forwarding graph requested by tenant c.
NFP^c	Set of forwarding paths requested by the tenant c.
K^c	Number of forwarding paths requested by the tenant c.
$NCT^c = (N^c, E^c)$	Request graph of tenant c.
n_i^c	Requested service node $\in N^c$.
$cpu(n_i^c)$	Required processing capacity of n_i^c.
$e^c(i,j)$	Logical link $\in E^c$ connecting nodes n_i^c and n_j^c.
$bw(e^c(i,j))$	Required bandwidth capacity of $e^c(i,j)$.
n	Substrate graph size.
m^c	Size of the request graph RG^c.

TABLE II: Notations

Hence, our eigendecomposition approach for VNF placement and Chaining should also achieve a trade-off between speed and efficiency and should scale with problem size.

V. Eigendecomposition for VNF Placement and Chaining

To solve the problem of VNF Placement and Chaining, we propose **an eigendecomposition of the adjacency matrices** of the request and the hosting infrastructure graph This approach uses an analytical method to obtain efficient solutions for the weighted graph matching problem. Our solution extends and adapts Umeyama's eigendecomposition approach [28] for this optimal matching between weighted graphs. This state of the art method is based on the eigendecomposition of the adjacency matrices of the input graph and the reference graph (graph to host the request) and on the Hungarian method [29] to extract mapping results. Details on Umeyama's solution for the weighted graph matching problem (WGMP) and our proposed Eigendecomposition approach for VNF Placement and Chaining are provided in the next sections.

A. *Umeyama's eigendecomposition approach for WGMP*

The problem of weighted graph matching which consists in finding the optimum matching between two weighted graphs (weights at each arc) was solved using an approximate solution in [28]. WGMP is solved by finding a mapping function ϕ between the nodes of the considered graphs while minimizing a similarity distance metric between them. Umeyama employs a spectral approach that provides efficient matching by using the eigendecompositions of the adjacency matrices of the graphs and the Hungarian method [29] to extract mapping results. An almost optimal matching can be found when the graphs are sufficiently close to each other. A brief description of the approach is provided below:

- To match two graphs G and H, we first construct their adjacency matrices A_G and A_H.
- The optimum matching between A_G and A_H consists in finding a permutation matrix P (containing matching results) that minimizes the similarity distance between the graphs G and H defined as $J(P)$. Hence, the aim is to find a solution with $J(P)$ very close to optimal using a matching algorithm that determines the permutation matrix.
- The permutation Matrix P is obtained by applying the Hungarian algorithm [29] (a well-known combinatorial optimization method) that maximizes the total weight on matrix $M = \bar{U}_H \bar{U}_G^T$, where U_G and U_H are the orthogonal/eigenvector matrices of A_G and A_H.

This approach presents some limitations if applied "as is" to VNFs Placement and Chaining in the Cloud. So, we have extended and adapted it to achieve joint node and link mapping and improve scalability compared to the current state of the art on SFC placement. Limitations and extensions are briefly discussed by summarizing the approach in [28] and presenting each time our introduced extension to make the method applicable to VNF placement and chaining:

- Umemaya's approach requires a pair of graphs of the **same size** and performs **only link mapping**:
  We have adapted the approach to support matching graphs of different sizes (since in SFC the request and substrate graphs do not have the same size) and to deal not only with link mapping but also with node mapping. More details are given in the next subsection.
- **Mapping is one to one** (a substrate resource hosts a single requested resource but this does not match with the Virtualization concept):
  We have adapted the approach to allow different VNF nodes/links to be hosted on the same physical node/link. An important extension is also to allow mapping virtual links into physical paths unlike Umemaya's approach where each single virtual link is mapped on exactly one physical link.
- The closest match is the best match (obtained using a squared similarity distance but this **violates the maximum capacity limits** in cloud resources, retains unfeasible solutions and misses feasible ones by stopping the exploration):
  An important extension is the design of a new matching algorithm for extracting appropriate mapping results achieving efficient VNF placement and steering traffic while respecting resource usage constraints.
- Umemaya's approach is adequate for graphs that are sufficiently close to each other (**ideally isomorphic**):
  As Umemaya's approach deals with graph similarity, matching results are better (or similarity distance is minimum) when the graphs are close to each other. In fact, the matching in [28] is different from the resource mapping considered in our problem because we care about resource availability and not about graph similarity. We have adapted the request and substrate graphs to bring them closer to each other by adjusting their size.

B. Eigendecomposition for VNF Placement and Chaining

We cast the VNF Placement and Chaining problem in networked cloud infrastructures into the weighted graph matching problem (WGMP). If we consider the substrate graph SG and a the VNF forwarding graph $VNF-FG^c$ of tenant c (formulated as a request graph NCT^c), WGMP is the problem of finding a one to one mapping function ϕ between N^s and N^c which minimizes a difference/distance criterion between SG and NCT^c. Our proposed solution for VNF Placement and Chaining in the Clouds extends and adapts Umeyama's eigendecomposition approach [28] for optimal matching between weighted graphs.

*1) **Substrate and request graphs**:* Let A_{SG} and A_{NCT^c} be the adjacency matrices corresponding to the substrate graph SG and to the NCT graph (request graph) NCT^c, respectively.

$$A_{SG} = \begin{cases} a^s_{i,j} = bw(e^s(i,j)) & \text{i} \neq \text{j} \\ a^s_{i,i} = cpu(n^s_i) \end{cases} \quad (1)$$

$$A_{NCT^c} = \begin{cases} a^c_{i,j} = bw(e^c(i,j)) & \text{i} \neq \text{j} \\ a^c_{i,i} = cpu(n^c_i) \end{cases} \quad (2)$$

Since WGMP deals with graphs of the same size, we add dummy isolated vertices to the request graph in order to make the graphs of equal size. So we transform each matrix A_{NCT^c} of size m^c into a matrix of size n by adding rows and columns containing only zeros.

*2) **Distance criterion**:* Let $J(\phi)$ be a criterion to measure difference/distance between substrate and request graphs [28], defined as:

$$J(\phi) = \sum_{i=1}^{n}\sum_{j=1}^{n}(a^s_{i,j} - a^c_{\phi(i),\phi(j)})^2 \quad (3)$$

If we consider A_{SG} and A_{NCT^c} the adjacency matrices of SG and NCT^c respectively, the $J(\phi)$ criterion can be reformulated by using a permutation matrix P that represents the node mapping function ϕ as follows [28]:

$$J(P) = \|PA_{SG}P^T - A_{NCT^c}\|^2 \quad (4)$$

The optimum matching between A_{SG} and A_{NCT^c} is hence reduced to the problem of finding the permutation matrix P that minimizes $J(P)$. In our work we compute the $J(P)$ value through a matching algorithm that determines the permutation matrix.

*3) **Eigendecomposition**:* As permutation matrices are orthogonal, the problem of finding a permutation matrix minimizing $J(P)$ can be solved by extending the space of J to the space of orthogonal matrices Q that minimize $J(Q)$. Eigendecomposition of the matrices A_{SG} and A_{NCT^c} are hence used to find a solution.

Let U_{SG} and U_{NCT^c} be the orthogonal/eigenvector matrices of A_{SG} and A_{NCT^c}, respectively. As demonstrated in [28], the permutation matrix P can be determined as permutation matrix that maximizes the trace $tr(P^T\bar{U}_{NCT^c}\bar{U}^T_{SG})$ which represents an instance of the assignment (bipartite maximum weighted matching) problem (see equation 5).

$$tr(P^T\bar{U}_{NCT^c}\bar{U}^T_{SG}) \leq n \quad (5)$$

Therefore, P is obtained by applying a Matching algorithm that maximizes the total weight on the matrix $M = \bar{U}_{NCT^c}\bar{U}_{SG}^T$).

There exist many optimization algorithms to solve the maximum matching problem (e.g. the Hungarian algorithm [29], a well-known combinatorial optimization method). But, in this paper we propose a heuristic Cloud-oriented algorithm that finds maximum weight matching while respecting the constraints imposed by the request and the service chaining (e.g. placement constraints, resource capacity constraints, dependance between paths of the VNF-FG...).

*4) **VNF placement and chaining algorithm**:* In order to explain our proposed algorithm, an example of VNF placement and chaining is provided in Figure 5 where a VNF-FG is transformed into an NCT graph subsequently hosted on the substrate graph (NFVI). The weights on the nodes represent CPU capacities and the link weights indicate bandwidth capacities. Both substrate nodes and links can host multiple virtual nodes and links as long as they have enough available capacity. Each physical node and link has a limit in terms of resource that cannot be exceeded cumulatively when assigning (or mapping) virtual resources to (onto) hosts.

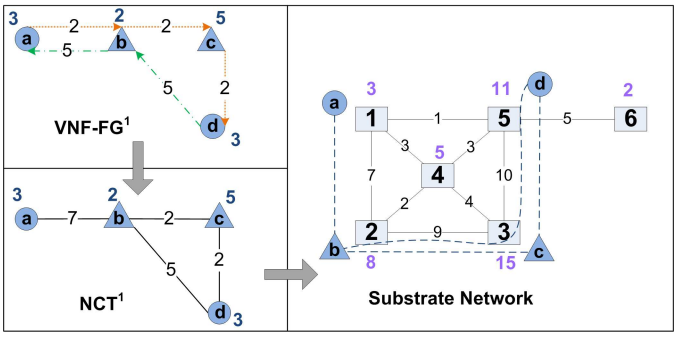

Fig. 5: An example of VNF placement and chaining problem

Our algorithm starts by computing the best paths between any two not directly connected substrate nodes (paths that maximize the minimum bandwidth along their route). The substrate graph adjacency matrix A_{SG} is updated next with weights equal to the calculated bandwidths. Information about paths is stored to be used when mapping Forwarding Paths.

Then, since the eigendecomposition deals with graphs of the same size, we add dummy isolated vertices to the request graph in order to make the graphs of equal size. So we transform each adjacency matrix A_{NCT^1} of size m^1 into a matrix of size n by adding $n - m^1$ rows and columns with zeros.

As depicted in Figure 5, the matrices described below correspond respectively to the substrate graph adjacency matrix A_{SG} and the request adjacency matrix A_{NCT^1}. In the example, matrix A_{NCT^1} of original size $m^1 = 4$ was padded with zero values to make both graphs of same cardinality (i.e., $n = 6$).

$$A_{NCT} = \begin{pmatrix} 3 & 7 & 0 & 0 \\ 7 & 2 & 2 & 5 \\ 0 & 2 & 5 & 2 \\ 0 & 5 & 2 & 3 \end{pmatrix} \Rightarrow \begin{pmatrix} 3 & 7 & 0 & 0 & 0 & 0 \\ 7 & 2 & 2 & 5 & 0 & 0 \\ 0 & 2 & 5 & 2 & 0 & 0 \\ 0 & 5 & 2 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$A_{SG} = \begin{pmatrix} 3 & 7 & \textcircled{0} & 3 & 1 & \textcircled{0} \\ 7 & 8 & 9 & 2 & \textcircled{0} & \textcircled{0} \\ \textcircled{0} & 9 & 15 & 4 & 10 & \textcircled{0} \\ 3 & 2 & 4 & 5 & 3 & \textcircled{0} \\ 1 & \textcircled{0} & 10 & 3 & 11 & 5 \\ \textcircled{0} & \textcircled{0} & \textcircled{0} & \textcircled{0} & 5 & 2 \end{pmatrix} \Rightarrow \begin{pmatrix} 3 & 7 & \textcircled{7} & 3 & 1 & \textcircled{5} \\ 7 & 8 & 9 & 2 & \textcircled{9} & \textcircled{5} \\ \textcircled{7} & 9 & 15 & 4 & 10 & \textcircled{5} \\ 3 & 2 & 4 & 5 & 3 & \textcircled{4} \\ 1 & \textcircled{9} & 10 & 3 & 11 & 5 \\ \textcircled{5} & \textcircled{5} & \textcircled{5} & \textcircled{4} & 5 & 2 \end{pmatrix}$$

As shown in Algorithm 1, starting from the extended adjacency matrix A_{NCT^c} (padded with zeros) and the adjacency matrix of the substrate graph A_{SG}, we derive the eigenvectors of these adjacency matrices, U_{SG} and U_{NCT^c}.

$$M = \begin{array}{c} \text{Substrate nodes} \\ \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 0.50 & 0.03 & 0.04 & 0.01 & 0.22 & 0.11 \\ 0.03 & 0.23 & 0.02 & 0.09 & 0.00 & 0.36 \\ 0.04 & 0.02 & 0.49 & 0.07 & 0.00 & 0.15 \\ 0.01 & 0.09 & 0.07 & 0.56 & 0.16 & 0.05 \\ 0.22 & 0.00 & 0.00 & 0.16 & 0.45 & 0.12 \\ 0.11 & 0.36 & 0.15 & 0.05 & 0.12 & 0.08 \end{pmatrix} \end{array} \begin{array}{c} \\ \\ a \\ b \\ c \\ d \end{array} \text{ VNF-FG nodes}$$

Fig. 6: Matrix M computation result for Figure5 example

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Algorithm 1 Eigendecomposition based approach

① Compute paths (in A_{SG}) that maximize the minimum bandwidth along the route between the not directly connected substrate nodes
② Extend A_{SG} with fictitious links whose weights correspond to the path bandwidths computed in Step ①
③ $U_{SG} \leftarrow EigenVectors(A_{SG})$
④ Aggregate the requested workloads and processing capabilities of VNF-FGs and their associated NFPs on the NCT^c graph
⑤ Extend A_{NCT^c} to have the same size as the A_{SG}
⑥ $U_{NCT^c} \leftarrow EigenVectors(A_{NCT^c})$
⑦ $M \leftarrow \bar{U}_{NCT^c} \times \bar{U}_{SG}^T$
⑧ $P \leftarrow$ VNF and VNF-FG Matching Algorithm(M)

The next step consists in building the matrix M and calling our proposed VNF and VNF-FG Matching Algorithm (Algorithm 2) to extract the solution (permutation matrix

P). This is illustrated in Figure 6 that highlights the potential candidate substrate nodes for hosting the VNFs and virtual nodes a, b, c, d as depicted: $Node1 \leftarrow a; Node2$ or $Node6 \leftarrow b; Node3 \leftarrow c; Node4$ or $Node5 \leftarrow d$. Note that this matrix M corresponds to the example of Figure 5. Matrix P derived by maximizing the trace $tr(P^T \bar{U}_H \bar{U}_G^T)$ finalizes the process by retaining only the nodes that meet the tenant's or consumer's graph request (more precisely the requested VNF-FG) in terms of nodes and links requirements. Indeed, the highest matrix M element values reflect the most likely best match, hosting node, for each VNF of the VNF-FG initial request. These most likely matches (highest values) have to be double checked by Algorithm 2 that verifies that all requirements at nodes, links and paths levels are met. For instance, the most likely candidate node for hosting requested $VNF\ b$ is $Node6$ (with value 0.36). Nevertheless, $Node6$ is not retained as there are no paths respecting the link requirement of 7. The maximum available bandwidth between $Node1$ and $Node6$ is only 5. The same can be said for candidate $Node4$ for hosting $VNF\ d$ and this despite its higher value in matrix M. Only candidate $Node5$ meets all the requirements by mapping the link between $VNF\ b$ and $VNF\ d$ via a path ($Node2 \rightarrow Node3 \rightarrow Node5$) with sufficient bandwidth.

The elements of matrix M are processed by Algorithm 2 to produce the mapping through our eigendecomposition approach (by setting the elements of P to 1 for each mapping found).

Algorithm 2 step 1 to step 3 find all candidate hosts from an eigendecomposition standpoint (elements with highest value in each row of M since they represent the best node matching) but these need to be verified in terms of ability to host before they are selected. Hence, steps 4 to 8 are essential to eliminate infeasible solutions and to not violate resource usage constraints.

If a candidate does not respect all established conditions, constraints and agreements, the algorithm returns to step 3 to process the next highest value in a row of matrix M. Steps 19 to 28 correspond to a function that is called by step 5 that verifies that "all links attached to the selected nodes and their neighboring nodes" meet the constraints and requirements before the permutation matrix P is returned as output of the mapping algorithm (via step 18). This output P contains the mapping results of VNFs and their associated forwarding paths onto the SG graph. It is essential to understand that the resource matching algorithm role is just to eliminate unfeasible solutions retained from the highest values in matrix M chosen in step 2. Actually, the eigendecomposition based approach in Algorithm 1 previously found jointly the candidate nodes and links in one shot. The solutions are in the candidate set but are finally extracted by the second algorithm.

The complexity of our solution is equal to the combined complexity of algorithms 1 and 2. The complexity of the eigendecomposition of adjacency matrices of NCT^c and SG depends directly on their size n and this complexity is known to be $O(n^3)$ [30]. The final step, step 8, of Algorithm 1, calls the VNF and VNF-FG Matching algorithm that has a complexity of $O(m^c \cdot n \cdot n - 1)$. Indeed, step 1 requires m^c iterations, the "Goto" step 3 from steps 11 and 15 involves n iterations and step 20 that verifies all the connection links to neighbours costs $(n-1)$ iterations in the case a full mesh graph request. Consequently, this leads to a worst case complexity of $O(n^3)$ overall for our solution.

Our approach is suboptimal because in step 7 of Algorithm 1 there is no specific order when running through the rows of matrix M and this produces few suboptimal solutions. The cost of ordering of the rows is increased complexity, that we would like to avoid at this stage. Note also that we do not need to do any backtracking in our approach since the derived eigen vectors and values (Algorithm 2) already provides appropriate hints in Matrix M to find the solution faster.

Algorithm 2 VNF and VNF-FG Matching Algorithm

Input: $NCT^c = (N^c, E^c)$;$SG = (N^s, E^s)$;$M = \bar{U}_{NCT^c} \times \bar{U}_{SG}^T$. **Output:** Matrix P: result of VNFs and VNF-FGs matching.

1 **for each** $n_i^c \in N^c$ **do**
2 $Row \leftarrow$ Get i^{th} row from M
3 $k \leftarrow$ Get index of highest value element from Row[1]
4 **if** $cpu(n_k^s) - cpu(n_i^c) \geq 0$ and (other node constraints are respected)[2] **then**
5 **if** $CheckLinks(n_i^c, n_k^s) = True$ **then**
6 $\phi(n_i^c) \leftarrow n_k^s$
7 $P(i,k) \leftarrow 1$
8 Update capacities on A_{SG}.
9 **else**
10 $Row[k] \leftarrow null$
11 **Goto** Step 3
12 **end if**
13 **else**
14 $Row[k] \leftarrow null$
15 **Goto** Step 3
16 **end if**
17 **end for**
18 **return** P

19 **function** CHECKLINKS(n_i^c,n_k^s)
20 **for each** $l \in neighbours(n_i^c)$ **do**
21 **if** l is mapped ($\phi(l) \neq null$) **then**
22 **if** $[bw(e^s(n_k^s, \phi(l))) - bw(e^c(n_i^c, l))] < 0$ and (other link constraints are respected)[3] **then**
23 **return** False
24 **end if**
25 **end if**
26 **end for**
27 **return** True
28 **end function**

1) To force nodes separation imposed by tenant requests, the algorithm would select the matrix M column index that corresponds to substrate nodes k that do not have a reservation yet and set the P matrix element to 1. To impose co-localization for virtual nodes in the same substrate node k (if required by the tenant), the algorithm will set elements of column k of matrix P to 1 for virtual nodes that must be co-localized.
2) memory, storage, locality, cost criteria and SLAs
3) latency, packet lost, cost criteria and SLAs

VI. Greedy algorithm for VNF Placement and Chaining

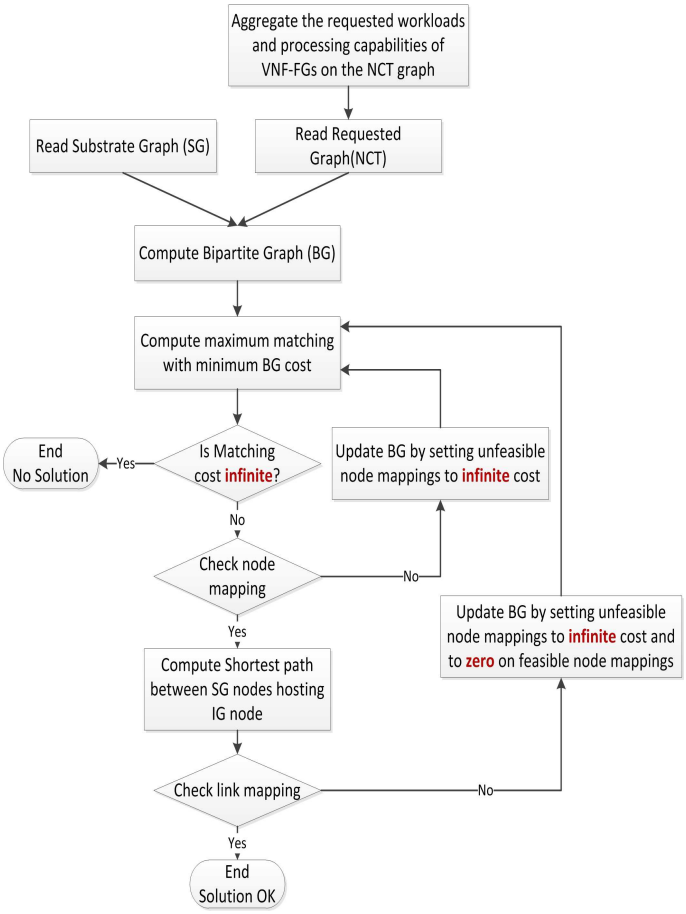

Fig. 7: Greedy algorithm

We propose a greedy algorithm (see Figure 7) for VNF placement and chaining to benchmark the eigendecomposition algorithm performance. The greedy solution is based on bipartite matching and can be summarized into 6 steps:

1) aggregate the requested workloads and processing capabilities of VNF-FG on the NCT^c graph;
2) create a complete bipartite graph (BG) based on the difference between the the NCT^c and SG nodes processing capacities;
3) compute maximum matching with minimum BG cost;
4) check node mapping: if node capacity and type constraints are respected, we move to link capacities checking. Otherwise, BG is updated by setting unfeasible node mappings to infinite cost. Then, the maximum matching with minimum BG cost is re-calculated (by going back to step 3);
5) compute best paths (paths that maximize the minimum bandwidth along their route) between SG nodes hosting NCT^c nodes;
6) check link mapping: if links capacities are respected, we return the mapping solution. Otherwise, BG is updated by setting unfeasible node mappings to infinite cost (since links depend on their associated nodes) and to zero on feasible node mappings. Then, the maximum matching with minimum BG cost is re-calculated (by going back to step 3);

VII. Performance Evaluation

This section describes our simulation settings and presents the results of a performance evaluation and a comparison of our eigendecompsitiong based heuristic (for VNF Placement and Chaining) with the greedy algorithm and the Multi-Stage algorithm proposed in [17].

A. Simulation Settings

To evaluate our approach, we run simulations using realistic topologies and parameters. We used the same conditions and parameter settings of simulation scenarios in the literature to obtain meaningful comparisons [17]. The algorithms were evaluated using a 2.70 GHz Quad Core server with 32 GBytes of available RAM. The GT-ITM tool was used to randomly generate substrate graphs as well as Network Forwarding Paths that are aggregated to form the requests (NCT). We assume that requests arrive according to a poisson process with an average rate of 5 requests per 100 time units and the lifetime of each request follows an exponential distribution with a mean of 1000 time units. We present results averaged over all simulation experiments (each value is an average over 100 instances) with a confidence interval of 95%. **Substrate Infrastructure.** As depicted in Table III, the considered substrate sizes vary between 100 and 5000 nodes with high connectivity percentage (50, 80 and 100 %). The available CPU capacity per service node and available bandwidth capacity per link randomly drawn in the 50 to 100 range. **Request: VNF-FG.** VNF forwarding graph are randomly generated then aggregated to form NCT graphs. The number of VNFs per request varies from 5 to 200 with 50 percent connectivity. The requested CPU capacity of each VNF is randomly drawn in the 0 to 20 interval and the requested bandwidth in the 0 and 50 range (see Table III).

	Substrate (Random Graph)
Size	[100-**5000**]
Connectivity	$P_{SG} = 0.5, 0.8, 1$
Nodes(CPU)	[50-100]
Links(BW)	[50-100]
	Request (Random Graph)
Size	[5-200]
Connectivity	$P_{NCT} = 0.5$
Nodes(CPU)	[0-20]
Links(BW)	[0-50]

TABLE III: Simulation Settings

B. Simulation Results

To evaluate the effectiveness of our proposed Eigendecomposition based heuristic, we compare its performance against the basic greedy heuristic for VNF placement and chaining in terms of convergence time and acceptance rate. Our key observations are summarized next.

(1) Eigendecomposition heuristic is fast and scales: The first experiment consists in comparing our Eigendecomposition approach with the greedy heuristic algorithm for

substrate graphs with 100 servers. The request/VNF-FG size varies from 5 to 20 and the connectivity of both substrate and request graphs is fixed to 50%. As depicted in Figure 8, the greedy algorithm requires more than 1.5 minutes to solve the problem for VNF-FG size equal to 50. To give a more comprehensive comparison and better understand the behavior of both heuristics, we increase the sizes of the substrate and request graphs.

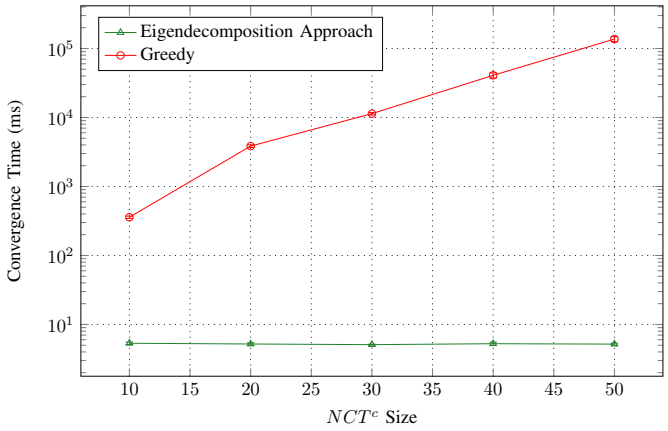

Fig. 8: Convergence Time (SG size = 100, P_{SG}=0.5, P_{NCT^c}=0.5)

In Figure 9, the substrate graph size is fixed to 1000 and the request size is varied from 10 to 50. This experiment shows that our Eigendecomposition heuristic manifests "flat" convergence time (less than 1.4 s for SG=1000), regardless of request size. On the other hand, the greedy heuristic needs more time to converge when increasing request size (more than 2 min for NCT=50). This is explained by the fact that the eigendecomposition heuristic is based on an analytical approach and depends only on substrate graph size, unlike, the greedy heuristic which is based on an iterative approach.

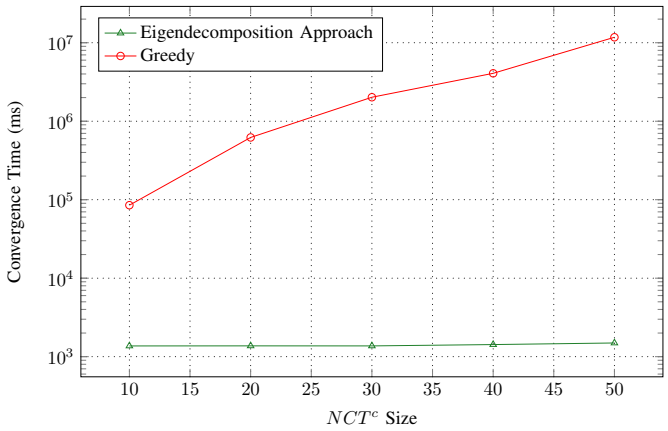

Fig. 9: Convergence Time (SG size = 1000, P_{SG}=0.5, P_{NCT^c}=0.5)

To strengthen the evaluation and to better understand the behaviour of the proposed algorithms (Eigen and Greedy), we have also compared our approach with the Multi-Stage algorithm in terms of convergence time as a function of SG and NCT graph sizes (see Figure 10 and Figure 11).

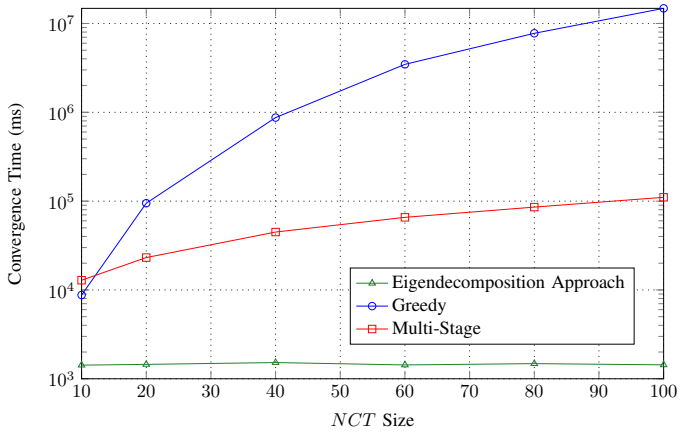

Fig. 10: Convergence Time when varying NCT size (SG size = 1000, $P_{SG} = 0.5, P_{NCT^c}$=0.5)

In Figure 10, the SG size is fixed to 1000 nodes and we vary the NCT request size in the range [10, 100]. The Eigendecomposition heuristic finds mapping solutions very fast (in less than 1.4 seconds). As expected the convergence time remains the same when increasing the NCT size since the performance of the Eigendecomposition algorithm depends on the substrate (or infrastructure) size. Convergence time of the Multi-stage and the Greedy algorithms on the contrary increases with the requested service chains sizes. The time needed to find a solution reaches 110 seconds for the Multi-stage and 246 minutes for the Greedy algorithm for $NCT = 100$. This figure confirms that our proposed Eigendecomposition algorithm is scalable with NCT graph size. If substrate graph sizes increase, the eigendecomposition algorithm will take longer to find solutions or reject the demands with a performance bounded by the algorithm complexity that is $O(n^3)$ with n representing the size of the infrastructure. The greedy and the multi-stage algorithm (of [17]) have exponential complexity and thus suffer from combinatorial explosion.

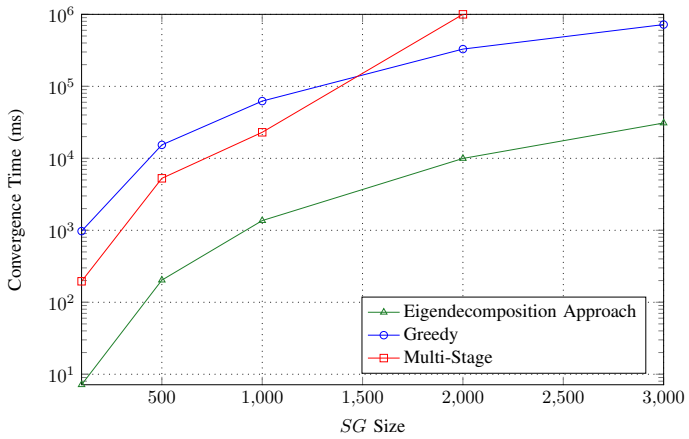

Fig. 11: Convergence Time when varying SG size (NCT size = 20, $P_{SG} = 0.5, P_{NCT^c}$=0.5)

Figure 11 shows that our Eigendecomposition approach is considerably faster compared with the Greedy and Multi-Stage algorithms when the NCT size is fixed to 20 (VNFs) and SG sizes vary between 100 and 3000 nodes. This Figure confirms results obtained in Figure 10 and shows that our Eigendecomposition algorithm is much faster (few seconds compared to tens to hundreds of seconds). For a SG size of 1000 nodes, the Eigendecomposition approach requires 1.36 seconds to compute the solution compared with 62 and 22 seconds required by the Greedy and the Multi-Stage algorithms respectively. Note that for substrate graph sizes greater than 2000 nodes, the Multi-Stage is unable to find a solution in reasonable time and takes unacceptable time before rejecting requests. The Eigendecomposition on the contrary can handle larger scale physical infrastructures in reasonable time.

(2) Eigendecomposition based heuristic is stable:

To further extend the evaluation of scalability, we run experiments for larger problem sizes. As depicted in Figure 12, we evaluate the convergence time for medium and large substrate graphs (over **5000** nodes) for the Eigendecomposition method. The flat behavior of the curves confirms that the convergence time depends only on the substrate graph sizes. Our Eigendecompsotion heuristic finds solutions in few seconds for substrate graph sizes equal or less than 3000. Even if convergence time is in the order of minutes for substrate sizes greater than 4000 (about 1 min for SG=4000 and 2 min for SG=5000), our heuristic computational complexity is quite acceptable compared with existing related algorithms which are limited to small substrate graphs [12] [3]. Note that the greedy and multi-stage approaches execution time increases with both the SG, NCT and VNF-FG graph sizes and are for this reason not included in Figure 12. The multi-stage of [17] handles only a forwarding path at a time (or simple chains) as opposed to the Eigendecomposition and the Greedy that treat entire VNF-FG graphs and thus handle multiple forwarding paths jointly and simultaneously. This also explain why the multi-stage performance is neither assessed nor reported in Figure 12 and Figure 13.

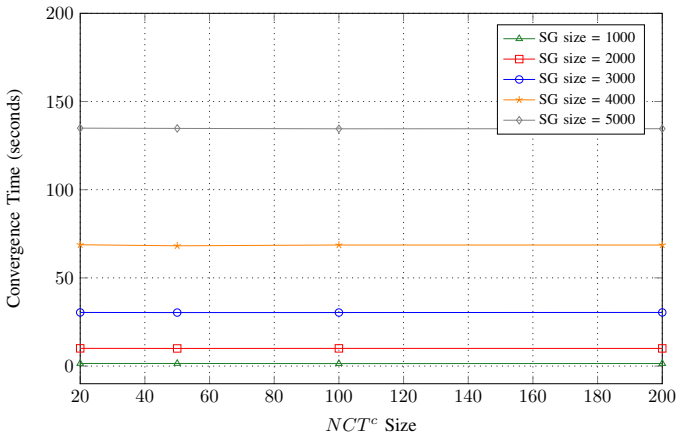

Fig. 12: Eigendecomposition Convergence Time

As depicted in Figure 13, our Eigendecomposition approach

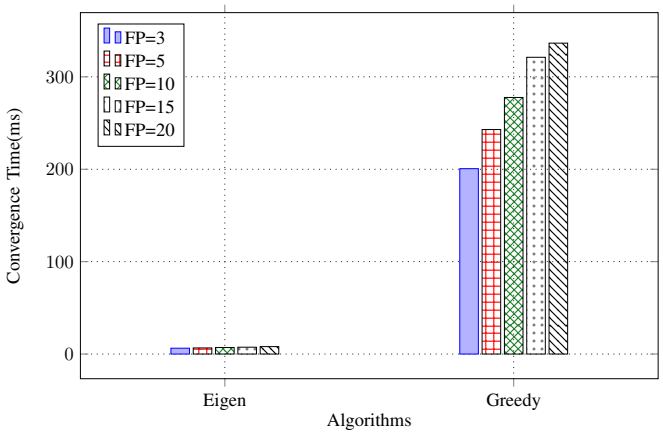

Fig. 13: Convergence time when varying the number of NFP per request

(SG size = 100, NCT^c size = 10)

is also stable and insensitive to the number of forwarding paths, unlike the greedy heuristic. In fact, iterative solutions are tightly dependent on all the parameters.

To emphasize this aspect, we also compare the convergence time of the eigendecomposition and the greedy heuristics with different requested resources types and proportions (% of virtual resources/VNFs vs % of physical resources/switches). We vary P the probability of having (or the percentage of) physical switches and PNFs in the VNF-FG request. The greedy algorithm consumes an important amount of exploration time to map required switches and PNFs to the substrate graph (see Figure 14) especially when the number of imposed physical switches is high since there are far fewer solutions available since the subspace to be searched is much smaller because of the induced hard node and link mapping constraints. Note that the proposed eigendecomposition approach in this work is fairly insensitive to these proportions of resource types in the substrate thanks to its natural capability of eliminating in very few iterations all unfeasible candidates and finding in one shot the requested mapping by processing the permutation matrix. Note that the multi-stage algorithm was not evaluated in Figures 13 and 14. The multi-stage algorithm does not appear in Figure 13 because it deals only with simple chains with only one FP (forwarding path) and in Figure 14 because it does not consider heterogeneous infrastructures with different physical node types (e.g. physical network function or PNF nodes taken on the contrary into account in the Eigendecompsotion and Greedy algorithms).

(3) Eigendecomposition based heuristic accepts more requests: Table IV highlights a characteristic of the eigendecomposition approach that converges faster and also rejects requests faster if the system is overloaded. As depicted in Figure 15, the eigendecomposition heuristic accepts more requests than the greedy algorithm that tends to use less efficiently the infrastructure resources by privileging certain nodes and thus reducing the likelihood of mapping future requests as the system load increases.

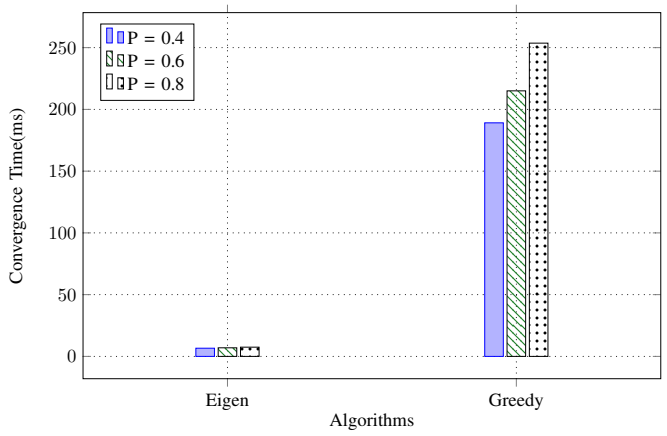

Fig. 14: Convergence time with different node types (SG size = 100, NCT^c size = 10)

The strength of our proposed eigendecomposition approach is its ability to find (as well as reject) in very few iterations (in Algorithm 2).Once matrix M has been computed (this step being the most time consuming, recall the $O(n^3)$ complexity), the algorithm is very fast since the previous steps (eigen vectors and values construction of the substrate and VNF-FG graphs), reveal the candidate subgraphs out of the entire substrate graph (NFVI) to explore for the solutions. Consequently the eigendecomposition method accelerates the search for the solution by transforming the exploration space and providing hints about the most relevant candidates.

	Eigen	Greedy
Convergence Time (ms)	4,93	236,56
Rejection Time (ms)	5	279,31

TABLE IV: Convergence and Rejection Time (SG size = 100, NCT^c size = 10, P_{NCT^c}=0.5)

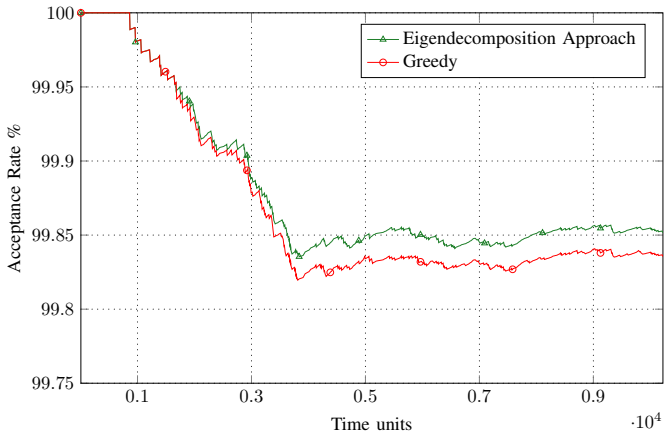

Fig. 15: Acceptance Rate (SG size = 100, NCT^c size =[2,10], $P_{SG} = 0.5$, P_{NCT^c}=0.5)

Figure 16 compares the Eigendecomposition approach with the Multi-Stage algorithm in terms of acceptance rate and provider revenue. The results show (in Figure 16a) that the proposed Eigendecomposition algorithm performs much better than the Multi-Stage algorithm in terms of acceptance rate since it accepts 99% of the VNF chaining requests while the Multi-Stage algorithm accepts only 20%. The computed revenue (Figure 16b) corresponds to the total revenue (number of accepted cpu and bandwidth resources multiplied by the associated revenue units) generated when accepting the NCT requests at time t. The Eigendecomposition algorithm increases the provider revenue by 77.75% compared with the Multi-Stage algorithm and confirms the results obtained by the acceptance rate performance in Figure 16a.

Because the acceptance rates of the eigendecomposition and the greedy algorithms are relatively close to each other, they are reported in Figure 15 without the multi-stage whose acceptance rates are significantly different and distant from the other two algorithms. For visibility reasons, the acceptance rate of the Multi-Stage algorithm was reported in a separate Figure 16a with the Eigendecomposition (only).

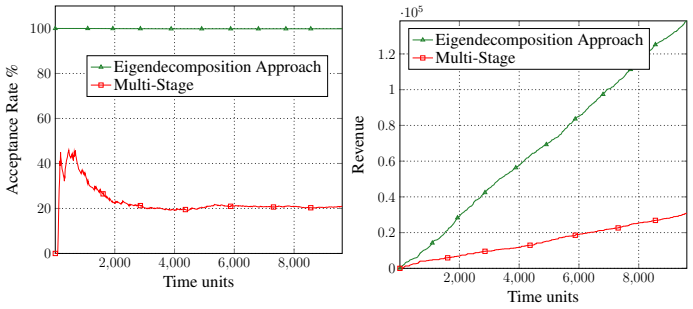

Fig. 16: Acceptance Rate and Provider Revenue

(4) Eigendecomposition based heuristic accepts more requests when requests are willing to wait:

Encouraged by the superior performance of the eigendecomposition method compared with the greedy algorithms, we introduce a waiting queue for the requests to take advantage of departures to repeat or persistently launch the algorithm when it does not find a solution on first attempt and would hence reject the request. Given the margin provided by the eigendecomposition algorithm, the additional delay can be tuned to be transparent to the consumer or to stay within the limit the consumer is willing to wait before receiving an allocation or be rejected. Introducing the queue can enhance the acceptance rate of requests when and if departures occur in the system.

The results are depicted in Figure 17 where variable R represents the number of authorized attempts before rejecting a request and parameter D is the number of departures the algorithm waits before it launches an optimization for finding solutions for the requests waiting in the queue. We can observe, as expected, that there is a tradeoff that needs to be achieved between R, D and the acceptance rate. The most favorable combination for these simulations is achieved by the curves $R3D3$ and $R3D10$ when the results are analyzed jointly with the summary provided in Table V.

The performance metric "waiting (%)" reported in Table V is the ratio of the waiting time in queue to the sojourn time

of accepted requests in the substrate. This is a measure of user/tenant satisfaction when their tolerance to delay before getting a service is used to improve the acceptance rate.

The results are respectively a waiting time of $8,6\%$ and 117 request rejected (instead of 148 for $Eigen$ or $EigenR0D0$) and $32,7\%$ for the waiting time and only 80 requests rejected. Note also that the average number of reattempts to find a solution remains quite low, less than 2 attempts. Depending upon the tolerance or willingness to wait for the consumer, the rejection rate can be reduced even further to 46 of course at the expense of delay (actually that increases to $67,32\%$ for $R10D10$).

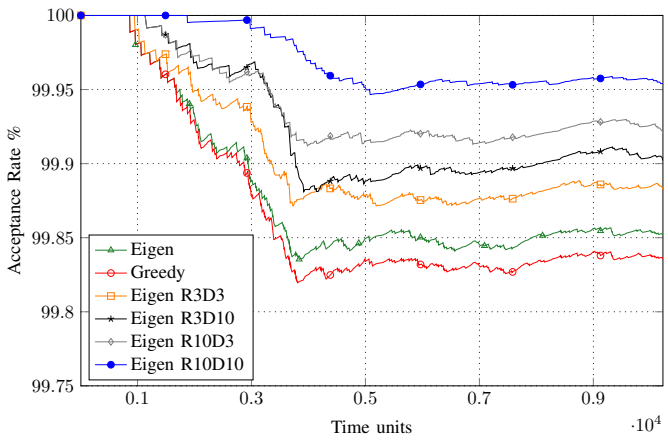

Fig. 17: Acceptance Rate (SG size = 100, NCT^c size =[2,10], $P_{SG} = 0.5, P_{NCT^c}$=0.5)

	Rejected request	Waiting (%)	Repeat (%)
Eigen	148	0	0
Greedy	164	0	0
Eigen R3D3	117	8,6	1,69
Eigen R3D10	80	32,7	1,59
Eigen R10D3	98	24,29	3,99
Eigen R10D10	46	67,32	3,4

TABLE V: Convergence Time
(SG size = 100, NCT^c size = 10, P_{NCT^c}=0.5)

VIII. Conclusion

In this paper, we presented a new approach to address the joint placement of virtualized network functions (VNFs) and their associated chains over distributed cloud environments. By relying on the eigendecomposition of the requested graph and the infrastructure graphs, we show that the placement of the VNFs and their chaining can be integrated in the adjacency matrices to find the closest mapping graph in the infrastructure that can respect the directed nature of VNF forwarding graphs (VNF-FG), their network paths and the dependence of these paths (and hence the traffic flows traversing the VNFs). The algorithm is shown to scale to thousands of nodes and links and to be insensitive to the number of requested VNF-FGs, the proportion of resource types and the connectivity in these input graphs. The algorithm has a fairly flat convergence time penalty governed by the size of the infrastructure graph. This stability in performance of the eigendecomposition makes this algorithm quite attractive compared to other conventional approaches that are very sensitive to the nature and proportions of resource types and to problem size such as the greedy algorithm presented also in this work. This robustness and stability and scaling capability of the algorithm encourage us to continue exploring how the eigendecomposition approach can be extended using a wider set of objectives and criteria to generalize the solution and address the multiple constraints, criteria and objectives encountered in NFV.

References

[1] R. Guerzoni, R. Trivisonno, I. Vaishnavi, Z. Despotovic, A. Hecker, S. Beker, and D. Soldani, "A novel approach to virtual networks embedding for SDN management and orchestration," in *2014 IEEE Network Operations and Management Symposium, NOMS 2014, Krakow, Poland, May 5-9, 2014*, 2014, pp. 1–7. [Online]. Available: http://dx.doi.org/10.1109/NOMS.2014.6838244

[2] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, Oct 2014, pp. 7–13.

[3] H. Moens and F. De Turck, "Vnf-p: A model for efficient placement of virtualized network functions," in *Network and Service Management (CNSM), 2014 10th International Conference on*, Nov 2014, pp. 418–423.

[4] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on*, Oct 2015, pp. 171–177.

[5] A. Gember, A. Krishnamurthy, S. S. John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar, "Stratos: A network-aware orchestration layer for middleboxes in the cloud," *CoRR*, vol. abs/1305.0209, 2013. [Online]. Available: http://arxiv.org/abs/1305.0209

[6] A. Mohammadkhan, S. Ghapani, G. Liu, W. Zhang, K. K. Ramakrishnan, and T. Wood, "Virtual function placement and traffic steering in flexible and dynamic software defined networks," in *2015 IEEE International Workshop on Local and Metropolitan Area Networks, LANMAN 2015, Beijing, China, April 22-24, 2015*, 2015, pp. 1–6. [Online]. Available: http://dx.doi.org/10.1109/LANMAN.2015.7114738

[7] X. Li and C. Qian, "The virtual network function placement problem," in *2015 IEEE Conference on Computer Communications Workshops, INFOCOM Workshops, Hong Kong, China, April 26 - May 1, 2015*, 2015, pp. 69–70. [Online]. Available: http://dx.doi.org/10.1109/INFCOMW.2015.7179347

[8] Y. Zhang et al., "Steering: A software-defined networking for inline service chaining," in *Proceedings of the 21st IEEE ICNP*, 2013, pp. 1–10.

[9] Z. Abbasi, M. Xia, M. Shirazipour, and A. Takcs, "An optimization case in support of next generation nfv deployment." in *HotCloud*, I. Ahmad and T. Kraska, Eds. USENIX Association, 2015. [Online]. Available: http://dblp.uni-trier.de/db/conf/hotcloud/hotcloud2015.html#AbbasiXST15

[10] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "Clickos and the art of network function virtualization," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 459–473. [Online]. Available: http://dl.acm.org/citation.cfm?id=2616448.2616491

[11] R. Riggio, T. M. Rasheed, and R. Narayanan, "Virtual network functions orchestration in enterprise wlans," in *IFIP/IEEE International Symposium on Integrated Network Management, IM 2015, Ottawa, ON, Canada, 11-15 May, 2015*, 2015, pp. 1220–1225. [Online]. Available: http://dx.doi.org/10.1109/INM.2015.7140470

[12] M. Luizelli, L. Bays, L. Buriol, M. Barcellos, and L. Gaspary, "Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, May 2015, pp. 98–106.

[13] S. Clayman, E. Maini, A. Galis, A. Manzalini, and N. Mazzocca, "The dynamic placement of virtual network functions," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, May 2014, pp. 1–9.

[14] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual Network Functions Placement and Routing Optimization," 2015. [Online]. Available: https://hal.inria.fr/hal-01170042

[15] W. Rankothge, J. Ma, F. Le, A. Russo, and J. Lobo, "Towards making network function virtualization a cloud computing service," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, May 2015, pp. 89–97.

[16] A. Abujoda and P. Papadimitriou, "MIDAS: middlebox discovery and selection for on-path flow processing," in *7th International Conference on Communication Systems and Networks, COMSNETS 2015, Bangalore, India, January 6-10, 2015*, 2015, pp. 1–8. [Online]. Available: http://dx.doi.org/10.1109/COMSNETS.2015.7098686

[17] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions in NFV," *CoRR*, vol. abs/1503.06377, 2015. [Online]. Available: http://arxiv.org/abs/1503.06377

[18] G. D. Forney, "The viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, March 1973.

[19] S. Sahhaf, W. Tavernier, M. Rost, S. Schmid, D. Colle, M. Pickavet, and P. Demeester, "Network service chaining with optimized network function embedding supporting service decompositions," *Computer Networks*, vol. 93, Part 3, pp. 492 – 505, 2015, cloud Networking and Communications {II}. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S138912861500359X

[20] ETSI GS NFV 003: "Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV".

[21] J. Halpern and C. Pignataro, "Service function chaining (sfc) architecture," Internet Requests for Comments, RFC Editor, RFC 7665, October 2015.

[22] ETSI GS NFV-MAN 001: "Network Functions Virtualisation (NFV); Management and Orchestration".

[23] P. Garg, P. Quinn, R. Manur, J. Guichard, S. Kumar, A. Chauhan, B. McConnell, M. Smith, C. Wright, U. Elzur, J. M. Halpern, W. Henderickx, T. Nadeau, S. Majee, D. T. Melman, K. Glavin, and P. Agarwal, "Network Service Header," Internet Engineering Task Force, Internet-Draft draft-quinn-sfc-nsh-07, Feb. 2015, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-quinn-sfc-nsh-07

[24] ETSI GS NFV 001: "Network Functions Virtualisation (NFV); Use Cases".

[25] W. Haeffner, J. Napper, M. Stiemerling, D. Lopez, and J. Uttaro, "Service function chaining use cases in mobile networks," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-sfc-use-case-mobility-05, October 2015, http://www.ietf.org/internet-drafts/draft-ietf-sfc-use-case-mobility-05.txt. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-sfc-use-case-mobility-05.txt

[26] Surendra, M. Tufail, S. Majee, C. Captari, and S. Homma, "Service function chaining use cases in data centers," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-sfc-dc-use-cases-04, January 2016, http://www.ietf.org/internet-drafts/draft-ietf-sfc-dc-use-cases-04.txt. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-sfc-dc-use-cases-04.txt

[27] VNF Orchestration For Automated Resiliency in Service Chains. draft-bernini-nfvrg-vnf-orchestration-00.

[28] S. Umeyama, "An eigendecomposition approach to weighted graph matching problems," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 10, no. 5, pp. 695–703, Sep 1988.

[29] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1982.

[30] C. G. F. Jacobi, "Concerning an easy process for solving equations occurring in the theory of secular disturbances," *J. Reine Angnew. Math*, 1846, 30:5194.

Marouen Mechtri has obtained an engineering degree and a master degree from National School of Computer Science, Tunisia and a Ph.D. from Telecom Sudparis, France, in 2014. He is currently a research fellow at the Telecom SudParis. He conducts his research within the Wireless Networks and Multimedia Services Department. His main research interests include network and Cloud computing resources optimization and emerging network paradigm such as SDN and NFV.

Chaima Ghribi received the Ph.D. Degree in Computer Science from Telecom SudParis, France, in 2014. She is currently a research fellow at the Telecom SudParis of Institut TELECOM in the Wireless Networks and Multimedia Services department. Her main research interests include Cloud computing, Distributed systems, Virtualization, Mathematical modelling and Algorithms.

Djamal Zeghlache Professor graduated from SMU in Dallas, Texas in 1987 with a Ph.D. in Electrical Engineering and joined the same year Cleveland State University as an Assistant Professor. In 1992 he joined the Networks and Services Department at Telecom SudParis of Institut Telecom where he currently acts as Professor and Head of the Wireless Networks and Multimedia Services Department. His current interests and research activities concern network architectures, protocols and interfaces to ensure smooth evolution towards loosely coupled future Internet, cloud networking and cloud architectures. He is currently addressing optimization, deployment, control and configuration challenges in cloud, SDN and NFV environments and systems.