NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

PROGRAM OF POSTGRADUATE STUDIES
”M.SC. IN COMPUTER SCIENCE”

DIPLOMA THESIS

Approximation Algorithms for Virtual Service Functions Chain Placement

Nikolaos S. Lazaropoulos

SUPERVISOR: V. Zissimopoulos, Professor NKUA

ATHENS

SEPTEMBER 2022

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
”ΠΛΗΡΟΦΟΡΙΚΗ”

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Προσεγγιστικοί Αλγόριθμοι για Ανάθεση Αλυσίδων Συναρτήσεων Εξυπηρέτησης με Περιορισμούς Διάταξης

Νικόλαος Σ. Λαζαρόπουλος

ΕΠΙΒΛΕΠΩΝ: Β. Ζησιμόπουλος, Καθηγητής ΕΚΠΑ

ΑΘΗΝΑ

ΣΕΠΤΕΜΒΡΙΟΣ 2022

DIPLOMA THESIS

Approximation Algorithms for Virtual Service Functions Chain Placement

Nikolaos S. Lazaropoulos
SN: CS3.20.0001

SUPERVISOR: V. Zissimopoulos, Professor NKUA

THREE-MEMBER EXAMINATION COMMITTEE

V. Zissimopoulos,
Professor NKUA

S. Kolliopoulos,
Professor NKUA

D. Fotakis,
Associate Professor NTUA

Examination Date: September 16, 2022

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Προσεγγιστικοί Αλγόριθμοι για Ανάθεση Αλυσίδων Συναρτήσεων Εξυπηρέτησης με Περιορισμούς Διάταξης

Νικόλαος Σ. Λαζαρόπουλος
ΑΜ: CS3.20.0001

Ημερομηνία Εξέτασης: 16 Σεπτεμβρίου 2022

ABSTRACT

Network Function Virtualization (NFV) is an emerging techonology in which network functions are no longer executed by proprietary software appliances but instead, can run on commodity servers located in distributed cloud nodes. These functions typically perform packet flow operations according to policies designed by network engineers. Examples of network functions include firewalls, load balancers, content filters, and deep packet inspection. This technology aims at dealing with the major challenges of today's enterprise middlebox infrastructure, such as monetary cost, capacity limitations, management complexity, energy consumption and failures. One of the main advantages of this approach is that Virtual Network Functions (VNFs) can be instantiated and scaled on demand without the need of installing new equipment. Network flows are often required to be processed by an ordered sequence of network functions. For instance, an Intrusion Detection System may need to inspect the packet before compression or encryption are performed. Moreover, different customers can have different requirements in terms of the sequence of network functions to be performed. The sequence of multiple VNFs required by network operators to perform traffic processing is called a Service Function Chain (SFC).

A virtual function can be executed on one or several servers. A fundamental problem arising when dealing with chains of network functions is how to map these functions to nodes (servers) in the network while achieving a specific objective. This problem is in general very hard to solve and most existing algorithms, mainly heuristics, have no provable performance guarantees.

For this reason, in this thesis we study notable approximation algorithms for SFC/VNF placement, targeting to highlight techniques of guaranteed approximation solutions. We aim to show how instances of the SFC/VNF placement problem reduce to known problems, like the Multiple Knapsack With Assignment Restrictions Problem and the Budgeted Maximum Coverage Problem. We also propose a problem definition that arises in cloud environments, where operators may need to route the most profitable service chains under resource constraints and we sketch a solution for a special instance of it.

SUBJECT AREA: Service Functions Chains, Virtual Network Functions

KEYWORDS: SFCs, VNFs, Approximation Algorithms

ΠΕΡΙΛΗΨΗ

Η εικονικοποίηση λειτουργιών δικτύου (NFV) είναι μια αναδυόμενη τεχνολογία στην οποία η επεξεργασία των ροών του δικτύου δεν εκτελείται πλέον από εξειδικευμένο hardware, αλλά αντίθετα, μπορεί να επιτελείται σε απλούστερους servers που βρίσκονται σε κόμβους ενός κατανεμημένου υπολογιστικού νέφους (cloud). Αυτές οι λειτουργίες συνήθως εκτελούνται σύμφωνα με πολιτικές που έχουν σχεδιαστεί από τους μηχανικούς του δικτύου. Τέτοιες λειτουργίες μπορεί να είναι τείχη προστασίας, εξισορροπητές φορτίου, φίλτρα περιεχομένου και βαθιά επιθεώρηση πακέτων. Αυτή η τεχνολογία στοχεύει στην αντιμετώπιση των βασικών προκλήσεων της δικτυακής υποδομής (data centers) των παρόχων υπηρεσιών, όπως το χρηματικό κόστος, οι περιορισμοί χωρητικότητας, η πολυπλοκότητα της διαχείρισης του δικτύου, η κατανάλωση ενέργειας και οι αστοχίες λογισμικού.

Ένα από τα κύρια πλεονεκτήματα αυτής της προσέγγισης είναι ότι οι λειτουργίες εικονικού δικτύου (VNF) μπορούν να δημιουργηθούν και να κλιμακωθούν κατ' απαίτηση χωρίς την ανάγκη εγκατάστασης νέου εξοπλισμού. Οι ροές δικτύου συχνά απαιτείται να υποβάλλονται σε επεξεργασία από μια διατεταγμένη ακολουθία λειτουργιών δικτύου. Για παράδειγμα, ένα σύστημα ανίχνευσης εισβολής μπορεί να χρειαστεί να επιθεωρήσει τα πακέτα πριν από τη συμπίεση ή κρυπτογράφηση τους. Επιπλέον, διαφορετικοί πελάτες μπορούν να έχουν διαφορετικές απαιτήσεις λειτουργίας σχετικά με την ακολουθία των λειτουργιών δικτύου που πρέπει να εκτελεστούν. Η διατεταγμένη ακολουθία πολλαπλών VNFs που απαιτείται να εκτελεστεί σε ένα δίκτυο ονομάζεται αλυσίδα εικονικών λειτουργιών (Service Function Chain - SFC).

Ένα θεμελιώδες πρόβλημα που προκύπτει όταν ασχολούμαστε με αλυσίδες λειτουργιών δικτύου είναι η ανάθεσή τους σε επιλεγμένους κόμβους του δικτύου με σκοπό την βελτιστοποίηση κάποιου κριτηρίου. Αυτό το πρόβλημα είναι γενικά πολύ δύσκολο να λυθεί και οι περισσότε-ροι υπάρχοντες αλγόριθμοι, κυρίως ευρετικοί, δεν έχουν εγγυημένη απόδοση.

Για το λόγο αυτό, στην παρούσα διπλωματική εργασία μελετάμε κάποιους σημαντικούς προσεγγιστικούς αλγόριθμους για την τοποθέτηση SFCs/VNFs. Προσπαθούμε να δείξουμε πώς ορισμένες περιπτώσεις του προβλήματος τοποθέτησης SFC/VNF μεταπίπτουν σε γνωστά προβλήματα, όπως το Πρόβλημα Πολλαπλών Σακιδίων με Περιορισμούς Ανάθεσης και το Πρόβλημα Μεγιστικού Καλύμματος με Περιορισμό Πόρων. Προτείνουμε επίσης έναν ορισμό προβλήματος που προκύπτει σε περιβάλλοντα υπολογιστικού νέφους. Εκεί οι διαχειριστές αποσκοπούν στις πιο προσοδοφόρες, από πλευράς ρυθμού μετάδοσης δεδομένων, αλυσίδες υπηρεσιών υπό περιορισμούς πόρων και προτείνουμε μια πιθανή λύση για μια ειδική περίπτωση του προβλήματος.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Service Functions Chains, Virtual Network Functions

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Αλυσίδες Εικονικών Λειτουργιών Δικτύου, VNFs, Προσεγγιστικοί Αλγόριθμοι

ACKNOWLEDGEMENTS

I would like to acknowledge and give my warmest thanks to my supervisor, Mr. V. Zissimopoulos, Professor of the Department of Informatics and Telecommunications of National and Kapodistrian University of Athens for the opportunity he provided me to deal with this specific topic, and for his guidance throughout the preparation of this thesis.

I would also like to thank the members of the Examination Committee: Mr. S. Kolliopoulos, Professor of the Department of Informatics and Telecommunications of National and Kapodistrian University of Athens and Mr. D.Fotakis, Associate Professor of the School of Electrical and Computer Engineering of National Technical University of Athens for honoring me through their presence in the examination of this thesis.

Also, I would like to express my gratitude to my family for their support during all these years of my studies. Their belief in me was what sustained me this far.

CONTENTS

LIST OF FIGURES

PREFACE

Network Functions Virtualization, NFV is an approach to telecommunications networking where the network entities that traditionally used dedicated hardware items are now replaced with computers on which software runs to provide the same functionality.

By running a network based around NFV, Network Functions Virtualization techniques, it is easier to expand and modify the network, and it is able to provide considerably more flexibility as well as being able to standardise on much of the hardware as it consists of additional computing power. In this way costs can be considerably reduced.

Working in various software companies for telecommunications, and living the developments around the transition of network solutions to the cloud, I noticed that there is almost no consistent way of making decisions for placing virtual applications in the servers. Each way is empirical, with its pros and cons. This fact piqued my interest to look for problem modeling techniques and algorithms that can provide direct and provable solutions in terms of their performance.

As in other fields, here too, theoretical computing can illuminate the search for guaranteed design approaches and help build efficient architectures in today's rapidly evolving cloud solutions.

1. INTRODUCTION TO NETWORK FUNCTIONS VIRTUALIZATION

This thesis studies approximation algorithms in the field of Network Functions placement and Service Function Chain placement/routing. We start with an introduction to Network Functions Virtualization, so that the reader can understand the technology field where the discussed approximation algorithms apply. We continue with a discussion around the Service Function Chain problem. After we study in a mathematical context existing techniques of guaranteed performance and finally we present a problem formulation inspired by ideas of cloud networking research.

1.1 Introduction

Network functions (NFs), also known as ”middleboxes”, are playing an increasingly important role in modern networks, ranging from mobile networks, enterprise networks, to datacenter networks. NFs improve the network performance (e.g., WAN Optimizer, web proxy and video transcoder, load balancer), enhance the security (e.g., firewall, IDS/IPS) or monitor the traffic (e.g., lawful interception, passive network monitor). Conventionally, NFs are built in dedicated hardware for performance concerns, which incur high capital investment and operating expense. Furthermore, they are hard to manage. Their replacement and upgrade involve non-trial human effort. In light of this situation, NFV was proposed [3] (Juliver Gil Herrera and Juan Felipe Botero, IEEE 2016) , aimed to address these issues by leveraging visualization technologies to consolidate NFs into general-purpose hardware platforms.
Network functions virtualization (NFV) is a new network architecture framework where network functions that traditionally used dedicated hardware (middleboxes or network appliances) are now implemented in software that runs on top of general purpose hardware such as high volume servers. NFV emerges as an initiative from the industry (network operators, carriers, and manufacturers) in order to increase the deployment flexibility and integration of new network services with increased agility within operator’s networks and to obtain significant reductions in operating and capital expenditures.
In November 2012, seven of the world’s leading telecom network operators selected the European Telecom Standards Institute (ETSI) [4] (R. Guerzoni et al., 2012) to be the home of the industry specification group for NFV. Under the paradigm of NFV, traditional middleboxes are managed as single modules of software, programmed to play the role of a particular Virtual Network Function (VNF), this allows modularity and isolation of each function, so they can be managed independently. In addition, NFV facilitates installation and deployment of VNFs on general purpose servers (e.g., x86-based blades), thus allowing dynamic migration of VNFs from one server to another, that is, to any place of the network.
In summary, NFV is the envisioned framework to solve most of the current network problems due to the wide use of specific hardware appliances. Also, it provides opportunities for network optimization and cost reduction. Moreover, it enables to configure hybrid sce-

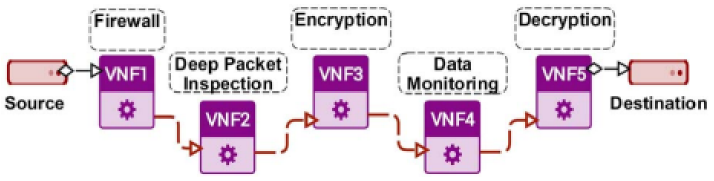

Figure 1.1: A Service Function Chain

narios where functions running on virtualized resources co-exist with those running on physical resources. Such hybrid scenarios may be important in the transition towards NFV. The traditional deployment of a Network Service (NS) requires the data traffic to pass through a certain fixed set of middleboxes in a particular order, which cause some processing according to the function they perform. The task of choosing the needed middleboxes and steer the traffic among them is commonly known as middleboxes orchestration [5] (A. Gemberet al., 2013). Traditionally, this task is performed manually, and is set at the forwarding table entries of routers; the above is a cumbersome and error prone process. Moreover, any placement of these physical middleboxes is destined to become ineffective over time; because it is very costly and impractical to keep changing the location of these hardware with changing network conditions. In the NFV ecosystem, a Network Service (NS) is a set of chained VNFs as shown in Figure 1.1. An NS is built and deployed in NFV by defining its: i) number of VNFs, ii) their respective order in the chain and iii) the allocation of the chain in the Network Functions Virtualization Infrastructure (NFVI), also called Substrate Network (SN). One of the main challenges to deploy NFV is to achieve fast, scalable and dynamic composition and allocation of NFs to execute an NS. However, since an NS requires a set of VNFs, achieving an efficient services' coordination and management in NFV raises two questions: 1) how to compose VNFs for a determined NS, and 2) how to efficiently allocate and schedule the VNFs of an NS onto a SN. The ETSI, through its NFV technologies group, is partnering with network operators and equipment vendors to promote NFV and is currently progressing with regard to the first question above.

1.2 VNFs Architecture

VNFs can be deployed and reassigned to share different physical and virtual resources of the infrastructure, so as to guarantee scalability and performance requirements. This allows Telecom Service Providers (TSPs) to rapidly deploy new and elastic services. In general, there are three main components in the NFV architecture: Services, NFVI and the NFV Management and Orchestration (NFV-MANO), as shown in Figure 1.2.

These components are described as follows. 1) Services: A service is a set of VNFs, that can be implemented in one or multiple virtual machines. In some situations, VNFs can run in virtual machines installed in operating systems or on the hardware directly; they are managed by native hypervisors or virtual machine monitors. A VNF is usually ad-

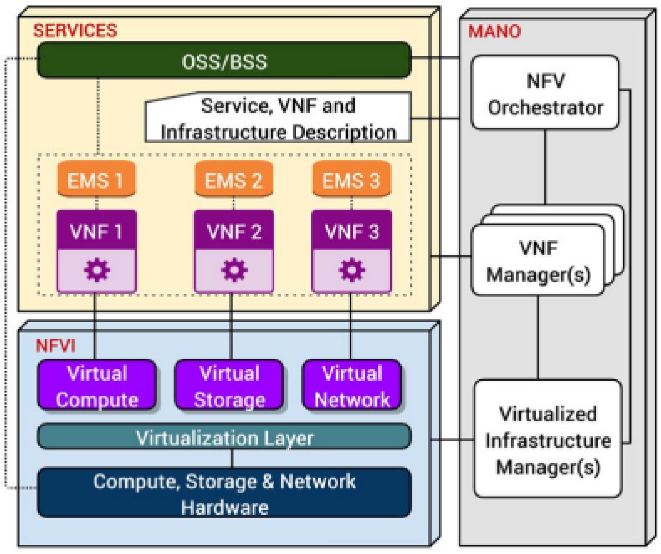

Figure 1.2: NFV Architecture

ministered by an Element Management System (EMS), responsible of its creation, configuration, monitoring, performance and security. An EMS provides the essential information required by the Operations Support System (OSS) in a TSP's environment. The OSS is the general management system, that, along with the Business Support System (BSS), help providers to deploy and manage several end-to-end telecommunications services (e.g., ordering, billing, renewals, problem troubleshooting, etc.). NFV specifications mainly focus on integration with existing OSS and BSS solutions. 2) NFVI: NFV infrastructure covers all hardware and software resources that comprise the NFV environment. NFVI includes network connectivity between locations, e.g., between data centers and the public or private hybrid clouds. Physical resources typically include computing, storage and network hardware providing processing, storage and connectivity for VNFs through the virtualization layer that sits just above the hardware and abstracts the physical resources (logically partitioned and assigned to VNFs). There is no specific solution for the deployment of NFV; rather NFV architecture can take advantage of an existing virtualization layer, such as a hypervisor, with standard features that simply extracts the hardware resources and assigns them to the VNFs. When this support is not available, often, the virtualization layer is achieved through an operating system that adds software on top of a non-virtualized server or by implementing a VNF as an application [1] (Xin Li and Chen Qian, CCNC, 2016). 3) NFV-MANO: NFV Management and Orchestration is composed of: the orchestrator, VNFs managers and Virtualized Infrastructure Managers. Such blocks provide the functionality required for the management tasks applied to the VNFs, e.g., provisioning and configuration. NFV-MANO includes the orchestration and lifecycle management of physical or virtual resources that support the infrastructure virtualization, and the lifecycle management of VNFs. It also includes databases that are used to store the information and data models defining both deployment as well as lifecycle properties of functions, services, and resources. NFV-MANO focuses on all virtualization-specific management tasks necessary in the NFV framework. In addition, the framework defines interfaces that can be used for communications between the different components of the NFV-MANO, as well as coordination with traditional network management systems (i.e., OSS and BSS) to allow the operation of both VNFs and functions running on legacy equipment. Summarizing, if an NS using a firewall and a DPI is deployed, then NFV-MANO shall be responsible to say where these VNFs are located on the physical network. In turn, these VNFs are controlled by the EMS and the same MANO. Besides, the virtualization layer exposes the physical resources of chosen NFVI locations to the VNFs.

1.3 Resource Allocation Problems in VNFs Placement

Resource allocation in NFV requires efficient algorithms to determine on which hypervisors VNFs are placed, and be able to migrate functions from one server to another for such objectives as load balancing, reduction of CAPEX and OPEX, energy saving, recovery from failures, etc. [2] (Juliver Gil Herrera and Juan Felipe Botero, 2016). In the NFV architecture framework the component that performs the resource allocation is the orchestrator.

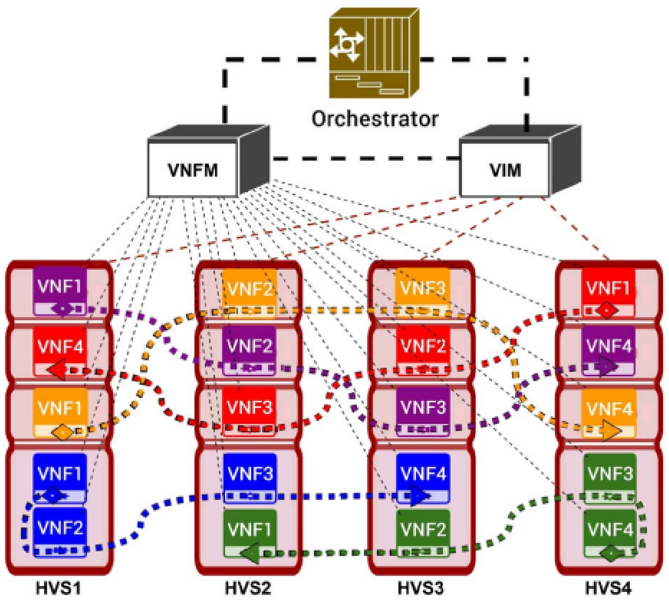

Figure 1.3: NFV Orchestrator

Figure 1.3 illustrates a scenario where the orchestrator manages VNFs through the VNF manager and the virtualized infrastructure manager (see Figure 1.2). The orchestrator evaluates all the conditions to perform the assignment of VNFs chains on the physical resources, leaning on the VNF managers and the virtualized infrastructure managers. The resource allocation in NFV is carried out in three stages: 1) VNFs Chain composition (VNFs-CC), also known in the literature as Service Function Chaining [6] (P. Quinn and J. Guichard, 2014) , 2) VNF Forwarding Graph Embedding (VNF-FGE) and 3) VNFs Scheduling (VNFs-SCH).

The three stages that conform the resource allocation problem in NFV-based network's architectures are presented below.

1.3.1 VNFs - Service Chain Composition (VNFs-SCC)

NFV exploits the flexibility introduced by virtualization to dynamically compose chains of VNFs and strategically deploy them on a set of physical network nodes so as to achieve a predefined operator's **objective** or to meet a Service Level Agreement (SLA), unlike the current static network function chain placement that depends on the physical location of the middleboxes in the SN. The ETSI defines an NS as entities composed by an ordered number of VNFs [2] (Juliver Gil Herrera and Juan Felipe Botero, 2016). That is, a packet must pass through a set of VNFs to be part of the offered network service. As VNFs are software, one of the main challenges that arises is: How to concatenate the different VNFs efficiently in order to compose an NS in the most adequate way, with respect to the TSP goals? This first challenge is the chaining process, that we call chain composition. TSPs will need to efficiently compose such chains to deploy customized and dynamic NFV-enabled network services.

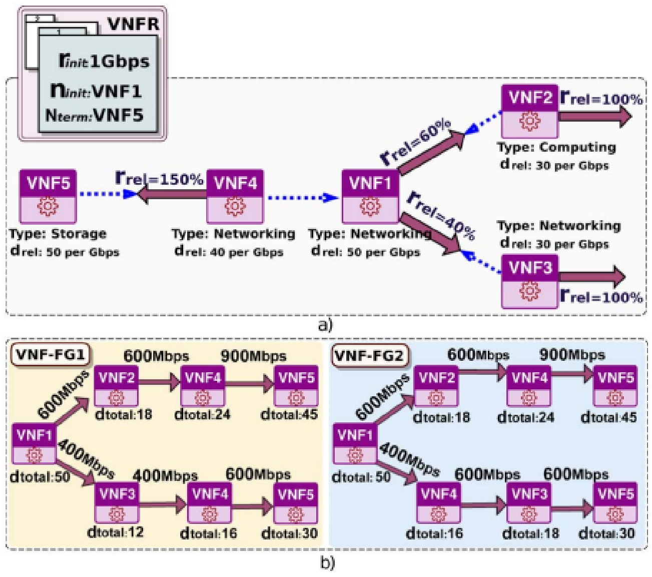

Figure 1.4: VNF Chain Composition

Figure 1.4 illustrates VNFs-CC. It shows a Virtual Network Functions Request (VNFR) and two possible chainings (VNF-FGs) of its VNF instances. For the i-th VNFR, VNFR^{i} , the initial data rate of the network flow -r$_{init}$(VNFR^{i})- and its VNFs are given. Some VNFs may split the traffic flow; for instance, a load balancer VNF separating incoming data into two streams can specify that 60% of the incoming traffic is forwarded to VNF2 and 40% to VNF3 (see VNF1, 2, and 3 in Fig. 1.4a). l^{i} (VNF) denotes the set of "outgoing" links of a VNF. For each link, relative traffic rate -r$_{rel}$- percentage with respect to the total outgoing VNF's traffic is defined (60% and 40% in the case of VNF1).

Neither bandwidth demands of the links nor capacity demands of the nodes in the VNF-FG are static; they depend on the ordering of the VNFs. This ordering is flexible, but it is tied to the dependencies between VNFs; i.e., the network flow first has to pass through a set of VNFs before it arrives at a specific VNF (blue dotted line in Fig. 1.4a from VNF4 to VNF1, implying that VNF4 depends on VNF1 and must therefore be executed after VNF1). Additionally, VNF link dependencies can be defined for VNFs that should selectively be placed on one of the sub-flows (see VNFs 2 and 3, both pointing to the VNF links of VNF1 in Fig. 1.4a).
Based on the dependencies, several valid chaining options (VNF-FGs) of VNFRs can be derived. Fig. 1.4b shows two possible VNF-FGs for the VNFR in Fig. 1.4a). In VNF-FG 1, the 1 GBps (r_{init}) is divided in 600 MBps (60%) to VNF2 and 400 MBps (40%) to VNF2 as indicated in the VNFR. In VNF-FG 2, the composition of the VNFs is different as in the lower branch VNF4 goes before VNF3, which is another perfectly feasible chaining solution (see Fig. 1.4b).

1.3.2 VNFs - Forwarding Graph Embedding (VNFs-FGE)

The chain of VNFs composing an end-to-end network service is called VNF-Forwarding Graph (FG). This resulting graph in the first stage is given as the input of the embedding stage. VNF-FG is composed by the ordered set of VNFs that the NS runs in order to fulfill service's attributes (e.g., reliability, availability, security and performance) [2] (Juliver Gil Herrera and Juan Felipe Botero, 2016).
VNF-Forwarding Graph Embeeding (FGE) is the second challenge identified in the NFV-RA, which seeks to find where to allocate the VNFs in the network infrastructure in a suitable way, considering a set of requested network services. Besides, resource optimization must be accomplished with regard to a specific objective (e.g., maximization of remaining network resources, minimization of SN's power consumption, optimization of a specific QoS metric, etc.). VNF-FGE can be seen as a generalization of the well-known Virtual Network Embeeding (VNE) Problem [2] (Juliver Gil Herrera and Juan Felipe Botero, 2016).
VNE is NP-hard [2] (Juliver Gil Herrera and Juan Felipe Botero, 2016) and, as VNF-FGE problem is a generalization of VNE, it is also NP-hard. Generally speaking, the problem consists on the mapping of virtual resources to candidate substrate resources. Only if all virtual resources can be mapped, the entire network is then embedded and substrate resources are actually spent. The two stages of VNE also apply to VNF-FGE: virtual node and virtual link mapping. In addition, in VNF-FGE, each VNF is annotated with a type: computing, storage or networking and therefore, it has to be allocated into a physical node that meet the VNF's type.

Figure 1.5 shows the deployment of an end-to-end network service $S = \{Firewall \rightarrow LoadBalancing \rightarrow Encryption \rightarrow PacketInspection \rightarrow Decryption\}$ between two substrate nodes in a NFV-enabled SN infrastructure. It illustrates the embedding stage involving: Orchestrator, service, virtualization layer and NFVI. The orchestrator is responsible of the embedding stage; it is in charge of the management and orchestration of software

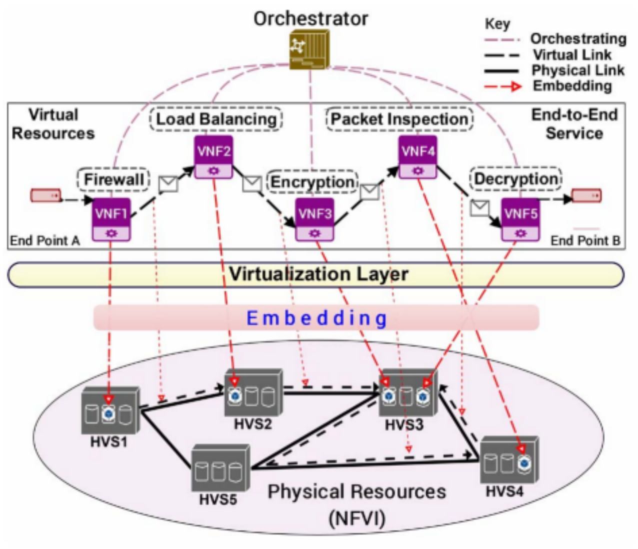

Figure 1.5: VNFs Forwarding Graph Embedding

resources and the virtualized hardware infrastructure to realize networking services. The service is composed by a set of VNFs that together provides a specific functionality. The virtualization layer abstracts the physical resources and anchors the VNFs to the virtualized infrastructure. It ensures that the VNF lifecycle is independent of the underlying hardware platforms by offering standardized interfaces. This type of functionality is typically provided in the forms of Virtual Machines (VMs) and their hypervisors which can be located in data centers, at network nodes, and in end-user facilities.
A High Volume Server (HVS) is considered a physical network node in a NFV based network architecture, which uses a hypervisor to manage virtual machines, according to the availability of resources (CPU, Disk, NIC and RAM). VMs running on top of HVSs can host one or more VNFs of the same type (computing, storage, networking).
An example to illustrate the embedding stage is shown in Figure 1.5. First of all, it is important to clarify that the orchestrator runs a VNF-FGE algorithm which makes embedding decisions, according to the objective to optimize. Here, as in the VNE, there are virtual node and link mapping phases. In the virtual node mapping phase; VNF1 is hosted onto HVS1, similarly, VNF2 is embedded onto HVS2, then, both VNF3 and VNF4 are mapped onto HVS3 and finally VNF5 is allocated onto HVS4. In the virtual link mapping phase, the algorithm maps each virtual link between VNFs to a path in the SN. It is important to note that the path may be composed of more than one physical link; for instance, the virtual link between VNF3 and VNF4 is mapped into the path composed of the physical links HVS3 - HVS5 and HVS5 - HVS4.

1.3.3 VNFs - Scheduling (VNFs-SCH)

A third and final stage of the NFV-RA problem is the scheduling process, that is called Virtual Network Functions Scheduling. This stage attempts to provide a functions' policy

of execution in order to minimize the total operating time without degrading the service performance and respecting all the precedences and dependencies between the VNFs composing the NS.
The NFV infrastructure is comprised of several and different HVSs, therefore, a proper scheduling of VNFs' execution should be performed in order to minimize the total execution time of the network services, and thus obtain improved performance.

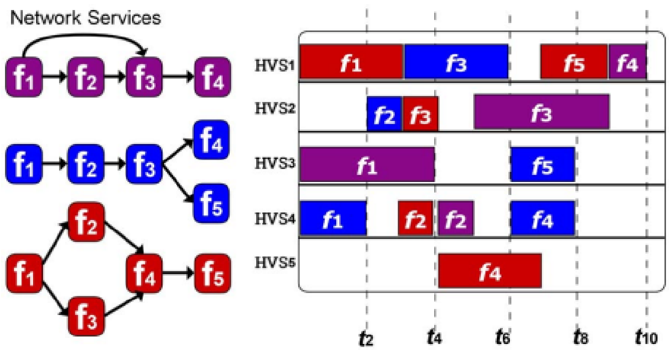

Figure 1.6: VNFs Scheduling

Figure 1.6 illustrates an example on how three different NSs, with different chains, and different network functions, can be scheduled over a limited NFVI, five servers in this case, minimizing the total execution time of the service set in order to maximize the system performance. Service 1 is composed of four VNFs; e.g., VNF1 runs onto HVS1 and takes 2.5 time units, VNF2 runs onto HVS4 and takes 1.0 time unit, VNF3 runs on HVS2 and takes 1.0 time unit, VNF4 runs onto HVS5 and takes 2.5 time units, VNF5 runs onto HVS1 and takes 2.0 time units. The runtime of service 1 was 10 time units. While service 2 has the shortest runtime. Table I shows the summary information for the services 1, 2 and 3 respectively.

2. THE SERVICE FUNCTION CHAIN PROBLEM

Service Function Chain (SFC) [11] (Weihan Chen et al., 2020) refers to connecting different network functions in specific sequence and providing corresponding service for users. The network functions in SFC are realized as different Virtualized Network Function (VNF). In actual network, SFC can be configured and adjusted according to different traffic demand. The configuration process involves two aspects: 1) **the placement of VNF** and 2) **the traffic steering (routing)** among different VNFs. In terms of VNF placement, the network operators (or Internet Service Providers) need to select the location for VNF Instance (VNFI), which can run VNF and allocate the resource (CPU, memory, etc.) for each VNFI. And in terms of traffic steering (routing), the path used to transmit traffic through specific VNFs of SFC needs to be determined. Proper SFC configuration can be helpful for improving network performance and reducing operational cost.
The VNF placement and routing optimization problem can be considered independently or jointly. When treating VNF placement optimization problem independently, VNF deployment and operational cost is considered as the prior optimization objective, the cost may include minimizing placement cost, minimizing traffic switching cost among different VNFs, etc. And the constraints of placement problem mainly focus on resource capacity constraints, which can be host CPU core number, link capacity or other network resources. In contrast, the optimization objective of routing problem tends to prioritize routing cost. It aims to find a path with least cost. The cost has many choices (such as financial cost, delay, QoS requirement, etc.). Meanwhile, the main constraint of routing problem is that user traffic flow should pass through the services provided by the SFC in the specified order.
In order to achieve better network performance, the VNF placement problem and traffic routing problem can be considered jointly. The optimization objective can be the combination of placement and routing optimization objectives. The constraints are also similar with the VNF placement optimization problem constraints plus routing constraints. However, optimizing VNF placement and routing jointly may cause some conflict. Because lower placement cost means less VNFIs are deployed, which results in higher routing cost (some traffic may be routed to longer path in order to achieve necessary network functions). On the contrary, to realize lower routing cost, more VNFIs need to be deployed, which causes placement cost increasing. Hence, finding a trade-off solution for joint optimization problem is necessary.

2.1 Virtual Network Function Placement

When a specific SFC is deployed, it first instantiates the required VNFs as VNFIs, and then places these VNFIs in proper location of the network. Different VNF placement schemes can affect the network performance and placement cost. For example, as shown in Figure 2.1a, if only one VNFI for each VNF of SFC is placed in the network, the placement cost (approximatively the number of deployed VNFIs) is minimized, but the network performance is relatively low. SFC traffic throughput is equal to the available bottleneck bandwidth of path shown in Figure 2.1a, which may not satisfy users requirement. However, if the placement scheme as shown in Figure 2.1b is adopted, the network performance can be better (traffic throughput can be improved), but the placement cost also ascends. During the placement process, network operators usually hope to allocate minimized resources to each VNFI while satisfying the performance requirements. VNF placement optimization can also bring financial benefit for network operators (e.g. reduction of deployment and operational cost).

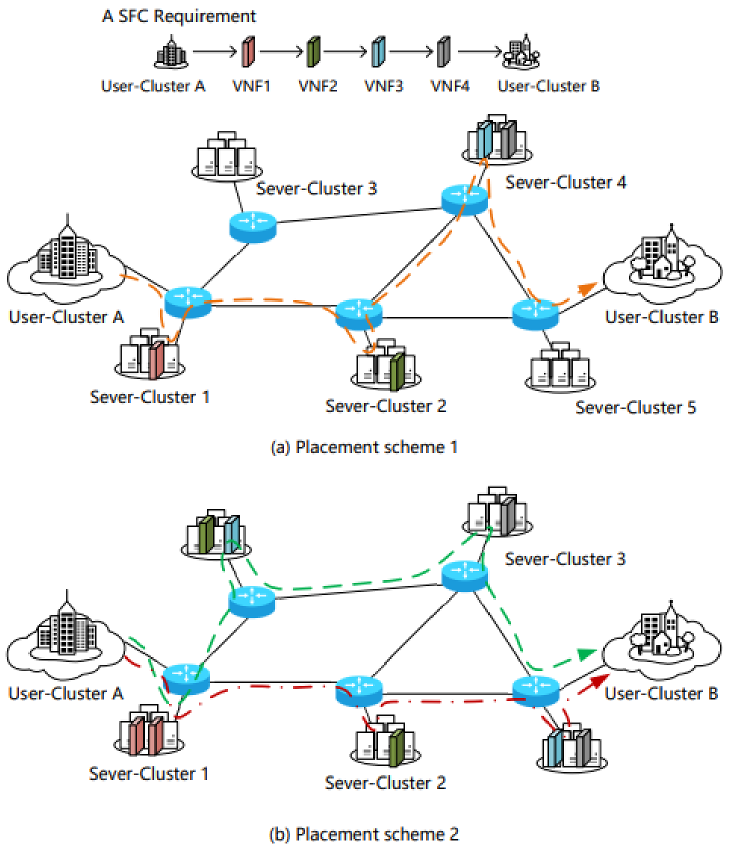

Figure 2.1: VNF Placement

In the current optimization solutions of VNF placement, the actual network is usually considered as a graph which includes a set of nodes and edges. The nodes are the abstract of forwarding devices in the network. Some of the nodes can connect with the server-clusters, and VNF can be deployed in these clusters. Each server-cluster has its own physical resources, containing CPU, memory, storage, etc. These resources should be allocated to the VNF as requirements. The edges in the graph represent the links between different nodes, and edges also have physical resource, mainly referring to link capacity. According to user requirements and resource constraints, the optimization solutions need to deploy VNFIs which are required by specific SFC in the graph, and then realize expected optimization goal.

In general, the cost that physical devices use to run VNF is mainly considered. This host resources allocation cost is related to the resource demand for each VNF and the number of VNFIs running on host, and the bandwidth resources allocation cost is related to the volume of traffic on each link.
Most optimization problems of VNF placement are modeled as Integer Programming problem or Mixed Integer Programming (MIP) problem [11] (Weihan Chen et al., 2020). Besides the optimization objective mentioned above, the problems also include the related resources and user demand constraints such as physical device capacity constraint, location constraint, link capacity constraint, throughput constraint and so on. These constraints are the boundary of VNF placement optimization problem, and they help to find optimal solution under specified conditions. Meanwhile, the computational complexity of solving the optimization problem also needs to be evaluated. Usually, the computational complexity is related to the number of nodes (namely physical devices that can run VNFs) in the network and the length of SFC (namely the number of VNFs in each SFC). More nodes or longer SFC means the computational complexity of solving process is higher.
Some VNF placement optimization problems are proved to be NP-hard [11] (Weihan Chen et al., 2020). That means it is difficult to realize fast solving for large-scale network. Therefore, some heuristic algorithms are proposed to realize fast solving. These heuristic algorithms include both classical algorithms (e.g. local search, greedy, etc.) and novel algorithms (e.g. bipartite graph matching, etc.).

2.2 Service Function Chain Routing For VNFs

Besides VNF placement, traffic routing also needs to be considered. The process of routing traffic requires to determine the forwarding path that traverses each VNF of SFC in specified order and consider the related network characteristics (such as link load, link transmission delay, etc.). The network operators usually wish to compute forwarding path efficiently and the routing cost could be minimized. In practice, traditional shortest path algorithm (e.g. Dijkstra's algorithm) can be helpful when computing forwarding path, but additional SFC constraints also need to be considered for satisfying user demands.
Similar to VNF placement optimization problem, SFC routing optimization problem also considers the actual network as a directed graph. The traffic should be transmitted from starting node to terminating node and pass through the VNFs of specified SFC. Meanwhile, the locations of these VNFs in the graph are assumed to be known in advance. The routing optimization solutions should calculate the shortest path with least cost and ensure the found paths are admissible.
The metric of SFC routing algorithm has many potential choices. It could be financial aspect (such as maintaining cost of forwarding devices, etc.) or network performance aspect (such as traffic propagation delay, user QoS demand, etc.). Existing optimization solutions usually aim to reduce the routing costs and improve the network performance like throughput. Minimization of the delay cost when calculating forwarding paths can be the only metric for link communication and VNF processing [11] (Weihan Chen et al., 2020). The reason is that delay is an important consideration in many networks, and it can also

be used to represent dynamic loads on network links and on VNF processing nodes.
The SFC routing algorithms need to find a forwarding path that can transfer traffic from source to destination with least cost. Meanwhile, they also need to ensure the traffic can be processed by required network services.

2.3 Joint Optimization of VNF Placement and SFC Routing

When VNF placement and SFC routing optimization problems are considered jointly, there cloud be a conflict between these two problems. For example, as shown in Figure 2.2a and 2.2b, there are three traffic requests T1 (from node 3 to 11), T2 (from node 11 to 1) and T3 (from node 10 to 5) demand SFC1 composed of VNF1, VNF2 and VNF3 (the order of VNFs is VNF1-VNF2-VNF3). In Figure 2.2a, if there is only one instance of SFC1 in the network, traffic flow T2 and T3 have to be routed over longer path, which causes more routing cost. However, if we deploy two SFC1 instances in the network, as shown in Figure 2.2b, the routing cost can be reduced due to using shorter forwarding paths. This example implies that optimizing VNF placement alone by instantiating fewer VNFIs may cause the traffic routing cost increasing. Whereas, if SFC routing optimization is considered preferentially, the additional VNF placement cost may be introduced, because more VNFIs are required to satisfy abundant traffic demand in today's network environment. Hence, joint optimization of VNF placement and SFC routing is necessary to find a trade-off optimal SFC deployment scheme.

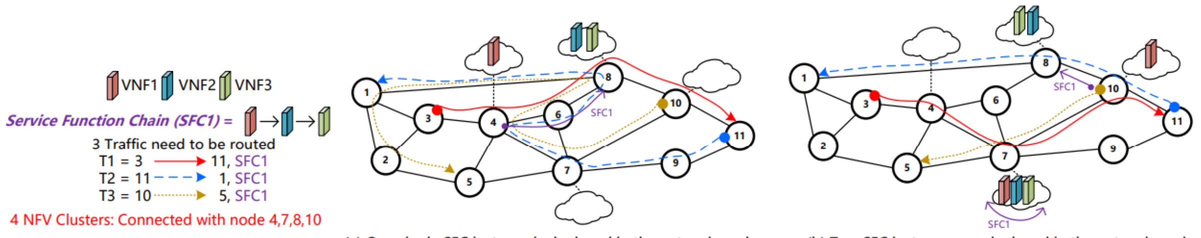
VNF1
VNF2
VNF3
Service Function Chain (SFC1) =
3 Traffic need to be routed
T1 = 3 → 11, SFC1
T2 = 11 → 1, SFC1
T3 = 10 → 5, SFC1
4 NFV Clusters: Connected with node 4,7,8,10
1
2
3
4
5
6
7
8
9
10
11
SFC1
(a) One single SFC instance is deployed in the network, and traffic T2 and T3 select longer paths
(b) Two SFC instances are deployed in the network, and traffic T2 and T3 select shorter paths

Figure 2.2: Conflict between VNF placement and SFC routing

Joint optimization solutions should deploy required VNFs of SFC properly, which means the deployment scheme can achieve high resource utilization or minimize the resources that need to be allocated with VNFs. Meanwhile, user traffic flow should also be routed through specified VNFs with QoS requirements. Besides these tasks, some solutions also consider the migration of VNFIs in response to the variation of user demand or network situation.
The objective of VNF placement and SFC routing joint optimization can be diverse. Some joint optimization solutions usually combine the VNF placement and SFC routing optimization objectives together. On the other hand, some solutions do not explicitly represent the VNF placement and SFC routing optimization objectives.
The type of optimization problem formulation mainly depends on the optimization objective. If the optimization objective is the combination of VNF placement and SFC routing

optimization objectives, the joint optimization problem is usually modeled as Mixed Integer Linear Programming (MILP) problem [11] (Weihan Chen et al., 2020). The reason is besides integer variables (like physical resources capacity), some SFC routing optimization solutions may involve real variables (like link delay). If the optimization objective does not involve real variables, the optimization solutions usually use ILP to model the optimization problem.
Since the joint optimization problems of VNF placement and SFC routing are basically NP-hard, most solutions propose corresponding heuristic algorithms to realize rapid solving. The details of each heuristic algorithm can be different according to the specific optimization problems. But the main idea of these heuristic algorithms is similar. They all rely on related network operational experience, leverage constraint relaxation, iteration and other methods to achieve the trade-off between optimality gap and computational complexity, and then find the result that is close to the optimal solution. However, the results solved by heuristic algorithm are usually near-optimal and the gap between near-optimal and optimal solutions cannot be estimated.

3. EXISTING APPROXIMATIONS

In this chapter we present notable approximation algorithms around SFC Placement and VNF Capacity Allocation problems.

3.1 Approximation algorithms for SFC Placement

Existing Service Function Chain placement algorithms can be roughly classified into two categories: ILP-based and greedy-based. These approaches typically have no provable performance guarantees. Although many works on SFC placement have been reported in the literature, the work in [7] (A. Tomassilli et al., IEEE INFOCOM 2018) is the first to propose a provably efficient algorithm to place chains of virtualized network functions within the network.
The authors model the network as a digraph $G = (V, E)$. A demand $d \in D$ is modeled by a **couple** composed of a path $path(d)$ of length $l(d)$ and a service function chain $sfc(d)$ of length $s(d)$. A path is a sequence of vertices in V . It is considered the case of an operator which has already routed its demands and which now wants to optimize the placement of network functions. A service function chain is an *ordered* sequence of functions in F, where F is the set of network functions. The flow associated with the demand should be processed by the network functions of its chain in the correct order. Each function $f \in F$ has a setup cost which may depend on the nodes. The setup cost of function f in node $v \in V$ is noted as $c(v, f)$.
The considered problem, referred to as SFC-PLACEMENT, is to find a *placement of network functions of minimum setup cost satisfying the service chain constraints of all demands.* It can be stated as follows.
Input: A digraph $G = (V, E)$, a set of functions F, and a collection D of demands. Each demand $d \in D$ is associated with a path $path(d) \in V^*$ and a sequence of functions $sfc(d) \in F^*$. Lastly, a cost $c : V \times F \to c(v, f)$, defining the cost of setting up the function f in node v.
Output: A *function placement* that is a subset $\Pi \subset V \times F$ of function locations, such that, all demands of D are satisfied. We say that a demand $d \in D$ associated with a path $path(d) = u_1, u_2, ..., u_{l(d)}$ and a chain $sfc(d) = r_1, r_2, ..., r_{s(d)}$ is satisfied by Π, if there exists a sequence of indices $i_1 \leq i_2 \leq ... \leq i_{s(d)}$, such that $(v_{i_j}, r_j) \in \Pi$, for $1 \leq j \leq s(d)$.
Objective: minimize $\sum_{(v,f)\in\Pi} c(v, f)$.

3.1.1 SFC with single function

The Minimum-Weight Hitting Set Problem (MIN-WHS), which is the hitting set formulation of the Minimum-Weight Set Cover Problem (MIN-WSC), can be formally defined as follows:
Input: Collection C of subsets of a finite set S.

Output: A hitting set for C, i.e., a subset $S^{'} \subseteq S$ such that $S^{'}$ contains at least one element from each subset in C.
Objective: Minimize the cost of the hitting set, i.e., $\sum_{x \in S^{'}} c_x$.

When all the demands have a service function chain which consists of a single function, the problem can be directly mapped to an instance of MIN-WHS:

1. the elements of S are the possible function locations, i.e., the vertices in V . Each element has cost $c(v)$.
2. the sets in C correspond to the paths of the demands in D. For each path $path(d)$, the corresponding set is the set of all the nodes in the path, i.e., $\{u_1, ..., u_{l(d)}\}$.

The placement of minimum cost covering all demands thus corresponds to a minimum cost hitting set. In the equivalent MIN-WSC formulation, the elements are the paths of the demands and the sets correspond to the function location for node v. The set associated with v has cost $c(v)$ and it is the set of all paths containing v. The equivalence directly gives us an $H(|D|)$-approximation using the greedy-algorithm for Set Cover [8] (V. Chvatal, 1979) on the positive side. On the negative side, it tells us that the SFC Placement Problem is hard to approximate within $ln|D|$ [9] (N. Alon et al., 2006).

3.1.2 Equivalence with Hitting Set

Even in the general case (with order), SFC Placement Problem is proven to be equivalent to MIN-WHS (and so to MIN-WSC) [7] (A. Tomassilli et al., IEEE INFOCOM 2018). For each demand $d \in D$, we denote with $l(d)$ and $s(d)$ the length of the associated path and chain respectively. Let $path(d) = u_1, u_2, ..., u_{l(d)}$ and assume that d requires the sequence of functions $sfc(d) = r_1, r_2, ..., r_{s(d)}$. Given a demand d, we build an *associated network* $H(d)$.

Definition 1. The network $H(d)$ is an associated one for a demand d if and only if it is built as follows:

1. $H(d)$ has $s(d)$ layers $L_1, L_2, ..., L_{s(d)}$. Each layer contains $l(d)$ nodes corresponding to the nodes of $path(d)$. We note (u_i, j) the i-th node of layer j.
2. There is an arc between the node (u, j) and the node $(v, j+1)$ if $u = v$ or if u precedes v in $path(d)$.
3. $H(d)$ has two other nodes, s_d and t_d. There is an arc between a node s_d and all the nodes of the first layer and an arc between all the nodes of the last layer and t_d.

Figure 3.1 shows the example of an associated network of a demand $d \in D$ routed on a path $path(d) = u_1, u_2, ..., u_{l(d)}$ that requires a chain $sfc(d) = r_1, r_2, ..., r_{s(d)}$. We then define

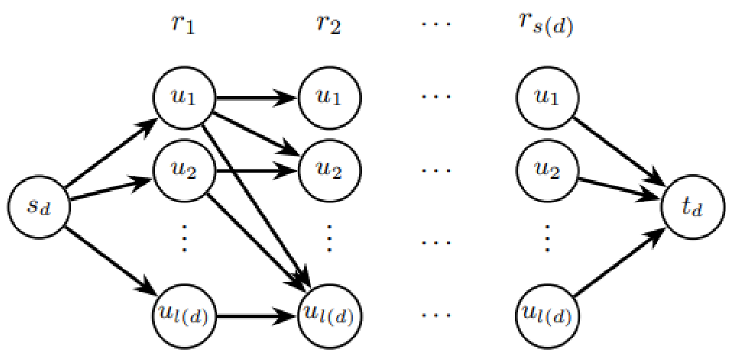

Figure 3.1: The Associated Network

the capacities to obtain the capacitated network $H(d, \Pi)$ associated with a demand d and a function placement Π: 1) all arcs have infinite capacity, and 2) each node has a capacity, and the capacity of the node u of layer i is 1 if $(u, r_i) \in \Pi$ and 0 otherwise.

Lemma 1. A demand $d \in D$ is satisfied by Π if and only if there exists a feasible st − path in the capacitated associated network $H(d, \Pi)$.
Proof. The intuition of the proof is that an $s_d t_d$ − path (or st − path in short) in the layered graph contains exactly one node from each layer and defines where the flow associated with the demand is going to be processed by the required functions in the specified order. Each layer is associated with a function - the j^{th} layer corresponds to the j^{th} function of the function chain $sfc(d) = r_1, r_2, ..., r_{s(d)}$. Since node (u, j) is connected to $(v, j+1)$ if and only if u precedes v in the path $path(d)$, the sequence of functions is performed in the right order when travelling along the path. Suppose there exists a feasible st − path, p. This means that there exists a set of indices $i_1, ..., i_{s(d)}$ such that $p = \{s, u_{i_1}, ..., u_{i_{s(d)}}, t\}$. This implies that the capacity of u_{i_j} is equal to one, i.e., $(u_{i_j}, r_j) \in \Pi$, for all $1 \leq j \leq s(d)$. Since, in the associated network $H(d, \Pi)$, node (u, j) is connected to $(v, j+1)$ if and only if u precedes v in $path(d)$, we have that $i_1 \leq ... \leq i_{s(d)}$. Therefore all functions of $sfc(d)$ are placed in the right order with respect to the nodes of $path(d)$, that is, d is satisfied by Π.
Suppose now that d is satisfied by Π. It means that there exists a set of indices i_1≤...≤$i_{s(d)}$, such that $(u_{i_j}, r_j) \in \Pi$, $\forall\ 1 \leq j \leq s(d)$. Nodes (u_{i_j}, j) of the associated network $H(d, \Pi)$ thus have capacity one. Moreover, there is an arc between (u_{i_j}, j) and $(u_{i_j+1}, j+1)$ as u_{i_j} precedes $u_{i_{j+1}}$ in $path(d)$. Hence, $\{s, (u_{i_1}, 1), ..., (u_{i_{s(d)}}, s(d)), t\}$ is a feasible st − path in $H(d, \Pi)$.
With this notion of associated network, the following problem is defined [7](A. Tomassilli et al., IEEE INFOCOM 2018):

Problem 1.HITTING-CUT-PROBLEM (D, c) is an instance of the Weighted Hitting Set problem where:

1. the elements are the function locations (u, f), for all $u \in V$ and $f \in F$. Its cost is $c(u, f)$.

2. the subsets of the universe correspond to all the st-vertex cuts of the associated networks $H(d)$ for all $d \in D$.

The problem is thus to find the sub-collection S of elements (functions placement) hitting all the subsets (cuts) of the universe of minimum cost.

Proposition 1. HITTING-CUT-PROBLEM (D, c) is equivalent to SFC-PLACEMENT (D, c).
Proof. By construction, a solution S of HITTING-CUT problem corresponds to a solution of SFC-PLACEMENT of same cost.
Let us show that S is feasible for HITTING-CUT-PROBLEM if and only if it is a feasible solution of SFC-PLACEMENT. The proof is direct using Menger's theorem for digraphs [10]. Consider a digraph and two vertices s and t not connected by an arc. The theorem states that the number of st − paths in a digraph is equal to the minimum st-vertex cut.
Lemma 1 says that all the demands in D are satisfied by Π if there exists an st − path in all the associated networks $H(d, \Pi)$ for each $d \in D$. We thus have that all demands are satisfied if all st − vertex − cuts of $H(P, \Pi)$ have a capacity larger or equal to one. Consider C an st-vertex cut. It is hit by S. This implies that in $H(d, \Pi)$, the capacity of the cut is larger than 1. This yields the proposition.

SFC Placement problem as defined in [7](A. Tomassilli et al., IEEE INFOCOM 2018), is thus equivalent to a Hitting Set Problem, for which approximation algorithms exist. However, the number of st-vertex cuts is exponential in the number of vertices of the digraph. To derive a polynomial algorithm, the authors try to reduce the size of an instance of CUT-HITTING-PROBLEM. To this end, they use the fact that checking only the *extremal* cuts is enough (An extremal cut is a cut that is not strictly included in another cut) and that, in their problem, the extremal cuts of the associated graphs have a specific shape that of *proper* st-cuts.

3.1.3 Greedy Algorithms for SFC Placement

The authors in [7](A. Tomassilli et al., IEEE INFOCOM 2018)propose two greedy algorithms to solve the SFC-PLACEMENT problem. Before stating them, they give the idea of *proper* st-cuts, which reduce the size of the search space for the HITTING-CUT problem which is equivalent to SFC-PLACEMENT problem.

Definition 2. A proper st-cut of the associated graph $H(d)$ is a cut of the following form:
$\{(u_1, 1), ..., (u_{j_1}, 1), (u_{j_1+1}, 2), ..., (u_{j_1+j_2}, 2), ..., (u_{j_1+j_2+...+j_{s(d)-1}+1}, s(d)), ..., (u_{l(d)=j_1+j_2+...+j_{s(d)}}, s(d))\}$
for $j_1, j_2, .., j_{s(d)} \geq 0$, such that $\sum_{i=1}^{s(d)} j_i = l(d)$. The right value of the above node pairs denotes the layer index in the associated graph.

Property 1. All the extremal cuts of the associated graphs are proper [7](A. Tomassilli et al., IEEE INFOCOM 2018) .
Proof. Let us consider a cut C in the associated graph. If it is possible to reach node (u_i, l)

from the source s, then node (u_{i+1}, l) can also be reached. Similarly, if the sink t can be reached from node (u_i, l), then the sink can also be reached from node (u_{i-1}, l). Suppose that there exists an extremal cut C such that, for a layer l, C contains nodes u_i, u_{i+2} with $u_{i+1} \notin C$. Since by definition C is a cut, we have 2 possibilities:

1. u_{i+1} at layer l cannot be reached by the source. Then, all the nodes u_j with $j \leq i+1$ in the layer $l-1$ cannot be reached, and so u_i is not reachable from the source. We can remove it from C and still get a cut. It follows that C is not an extremal cut (contradiction).

2. u_{i+1} at layer l cannot reach the sink. In the same way, u_{i+2} cannot reach the sink. We can then remove u_{i+2} from C and still get a cut. C is not an extremal cut (contradiction).

Therefore, for a layer l, an extremal cut C cannot contain nodes $(u_i, l), (u_{i+2}, l)$, if it does not contain (u_{i+1}, l).

As an example, consider a demand d that requires the service function chain $\{f1, f2\}$. Suppose that the demand is routed on the path $P = \{a, b, c\}$. There are four proper cuts: $\{(a,2),(b,2),(c,2)\}, \{(a,1),(b,2),(c,2)\}, \{(a,1),(b,1),(c,2)\}, \{(a,1),(b,1),(c,1)\}$ corresponding respectively to $j_1 = 0, ..., l(d)$.

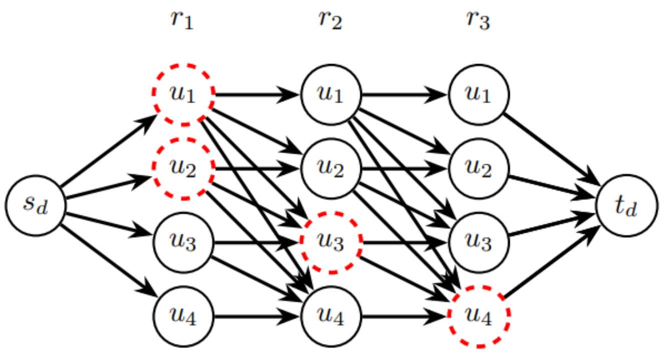

Figure 3.2: Example of a proper cut (dashed nodes in red) for the layered graph relative to a demand d associated with a path of length 4 and a chain of length 3

A new problem of smaller size is defined in [7](A. Tomassilli et al., IEEE INFOCOM 2018).

Problem 2. HITTING-PROPER-CUT-PROBLEM(D, c) is the same problem as HITTING-CUT-PROBLEM(D, c), except that the sets to be hit are only the proper st-vertex-cuts of the associated networks $H(d)$ for all $d \in D$.

Proposition 2. The problem SFC-PLACEMENT(D, c) is equivalent to a Hitting Set Problem with $\sum_{d \in D} \binom{l(d)+s(d)-1}{s(d)-1}$ sets as an input. If each demand requires at most s_{max} network functions and is associated with a path of length smaller than l_{max}, then the size of the instance is at most $O(|D| \cdot (l_{max})^{s_{max}-1}$.

Proof. HITTING-PROPER-CUT-PROBLEM(D, c) is equivalent to HITTING-CUT-PROBLEM(D, c) as it is enough to consider extremal sets of the collection in a Hitting Set Problem and all extremal cuts are proper cuts. SFC-PLACEMENT(D, c) thus is equivalent to HITTING-PROPER-CUT-PROBLEM(D, c).
The size of the ground set of HITTING-PROPER-CUT-PROBLEM(D, c) is the number of proper cuts of all the associated networks. For each path P, the number of proper cuts of $H(P)$ is simply equal to $\binom{l(d)+s(d)-1}{s(d)-1}$.
Indeed, to obtain the indices $j_1, ..., j_{s(d)}$ defining a proper cut, it is sufficient to select $s(d)-1$ numbers (notice that $(u_1, 1)$ and $(u_{s(d)}, s(d))$ are pre-selected nodes in any proper cut) between 0 and $l(d)+s(d)-1$. Without loss of generality, we call them $n_1 \leq ... \leq n_{s(d)-1}$. We then take $j_1 = n_1-1, j_i = n_i-n_{i-1}-1$ for $1 \leq i \leq s(d)-1$, and $j_{s(d)} = (l(d)-s(d)-1)-n_{s(d)-1}$. We have that $\sum_{i=1}^{l(d)} j_i = l(d)$, so the indices define a proper cut. There are $\binom{l(d)+s(d)-1}{s(d)-1}$ ways of choosing $s(d)-1$ elements in a set of $l(d)+s(d)-1$ elements. It yields the number of proper cuts. The size of the ground set is thus $\sum_{d \in D} l(d)^{s(d)-1}$.
Last, we have $\binom{l(d)+s(d)-1}{s(d)-1} = O(l(d)^{s(d)-1})$. This gives that the number of proper cuts over all paths of the set of demands D is of the order $O(|D|(l_{max})^{s_{max}-1})$.
Proposition 2 leads to two approximation algorithms, a greedy one and one using LP-rounding.

3.1.4 Naive and Faster Greedy Algorithms

Naive Greedy Algorithm. The naive greedy algorithm is just the classic greedy algorithm for set cover [8](V. Chvatal, 1979).It consists of a main loop: while there are proper cuts not hit, it selects the function location with the smallest average cost per newly hit proper cut. When the demands are routed on paths with length at most l_{max} and require at most s_{max} functions, the greedy algorithm achieves an approximation ratio equal to $H(|D|l_{max}^{s_{max}-1}) \sim ln(|D|) + (s_{max}-1)ln(l_{max})$ [8](V. Chvatal, 1979), where $H(n)$ is the n-th harmonic number.

Problem for large chains. When the number of functions in the service chains is large, the greedy algorithm could become impractical if it is implemented naively. In fact, the greedy algorithm selects the function location with the smallest average cost per newly hit proper cut. In a naive implementation, it is necessary to **generate explicitly all the proper cuts**, and this is not practical since, for a demand d, there may be $O(l_{max}^{s_{max}-1})$ of such cuts. Indeed, l_{max} is in the order of the network diameter. As an example, consider a network that has a diameter of 28. For a chain of length 10, we would have $\binom{37}{9}$ proper cuts. However, since the structure of the proper cuts is very specific, we can take advantage of it, providing a much faster greedy algorithm.

Faster Greedy Algorithm, SFCFastGreedy. The main idea of the faster greedy algorithm is to avoid generating all proper cuts by showing it is enough to keep track of the *number of not hit proper cuts*. We show here that, by using dynamic programming, this number can be counted in time $O(|D|l_{max}^2 s_{max})$ instead of $O(|D|l_{max}^{s_{max}})$.
The authors in [7](A. Tomassilli et al., IEEE INFOCOM 2018)introduce some useful notation.

For a demand $d = (path(d), sfc(d))$, a function placement Π can be seen as a matrix A_d with $l(d)$ rows and $s(d)$ columns and for which $A_d[i,j] = 1\ iff\ (u_i, r_j) \in \Pi$. We note $A_d[i:j, k:l]$ the submatrix of A_d considering only the rows from i to j and the columns from k to l.
For a demand $d = (path(d), sfc(d))$ and a function placement Π (or equivalently A_d), we note $N(d)$ the number of proper cuts not hit by A_d. It can be computed using the recursive function $N(r,c)$ defined below. We have $N(d) = N(l(d), s(d))$ with

$$\begin{array}{l} N(r,c) = 1_{i^*(r,c)=0} + \sum_{j_c=0}^{r-i^*(r,c)} N(n-j_c, c-1), c \geq 2 \\ N(r,1) = 1_{i^*(r,c)=0} \end{array} \tag{3.1}$$

where $i^*(r,c)$ is defined as follows. We consider the matrix $A_d[1:r, 1:c]$. We consider the ones placed in the last column of the matrix, column c. If there are none, $i^*(r,c) = 0$. Otherwise, $i^*(r,c)$ is the maximum index of such ones, that is, $i^*(r,c) = \max_{0 \leq i \leq l(d)}\{i | A_d[i,c] = 1\}$.
The explanation of the formula is the following. We carry out a recursion on the columns of $A_d[1:r, 1:c]$. First, if $i^*(r,c) = 0$, the cut $\{(u_1,c), ..., (u_r,c)\}$ is not a hit. We thus count $1_{i^*(r,c)=0}$. We then consider all possible values of j_c for the proper cuts (recall that a proper cut is defined by a set of indices $j_1, ..., j_c$). For a not hit proper cut, $j_c \leq l(d) - i^*$. For a possible value of j_c, the number of corresponding not hit proper cuts is equal to the number of not hit proper cuts in the submatrix $A_d[1:r-j_c, 1:c-1]$ for a path of length $r - j_c$ and a chain of size $c-1$, that is, $N(r-j_c, c-1, A_d[1:r-j_c, 1:c-1])$.
$N(r,c)$ can be computed using dynamic programming, see the function NC of Algorithm 1. We use a table T with r rows and c columns to keep track of the partial results of the computation. Initially, $T(i,1) = 1_{i^*(r,c)=0}$ for $1 \leq i \leq r$.

An example. Consider a demand d with $sfc(d) = f1, f2, f3$ and $path(d) = u1, u2, u3$. Let Π be a potential function placement. $\Pi = \{(u1, f1), (u3, f2), (u2, f3)\}$, that is, f_1 is installed on u_1, f_2 on u_3, and f_3 on u_2. All the required functions are placed, but not in the right order. We show that, in this case, some proper cuts of the associated network $H(d,\Pi)$ are not hit. $H(d,\Pi)$ has $\binom{5}{2} = 10$ proper cuts according to Proposition 2. We compute here the number of not hit proper cuts from this set without generating them. The matrix A_d associated with the demand and the starting table T in Algorithm 1 would be the following:

$$A_d = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, T = \begin{bmatrix} 0 & \_ & \_ \\ 0 & \_ & \_ \\ 0 & \_ & \_ \end{bmatrix}$$

As $A_d[1,1] = 1$, we have $i^*(3,1) = 1 \neq 0$ (the cut $\{(u_1,1), (u_2,1), (u_3,1)\}$ is hit). We thus initialize the first column of T with only zeroes.
From A_d we can see that $i^*(3,3) = 2$. In order to compute $T(3,3)$ the following steps are necessary:
$T(3,3) = T(1,2) + T(2,2) + T(3,2)$
$T(1,2) = 1 + T(1,1) = 1$

$T(2,2) = T(2,1) + T(1,1) + 1 = 1$
$T(3,2) = T(3,1) = 0$
Since $T(3,3) = 4$, we can derive that 2 proper cuts, out of the overall 10 proper cuts of $H(P,\Pi)$, are not hit. Note that this corresponds to the two proper cuts $\{(v1,f2),(v2,f2)(v3,f3)\}$ and $\{(v1,f2)(v2,f3)(v3,f3)\}$. This shows that the order of the functions is not valid.
From this approach, we can derive a faster algorithm with pseudo-code given in Algorithm 1. At each iteration, the algorithm selects the pair (u,f) of minimum cost, i.e., with the smallest average cost per newly hit proper cut. In order to do this, it makes use of the function NC, calling it for each demand and for each pair $(u,f) \in V \times F$. The pair of minimum cost is added to the solution Π. Then, the number of remaining proper cuts to be hit is updated. This process is repeated until all the proper cuts are hit.

Algorithm Complexity. The number of iterations of the main loop of the algorithm is bounded by $|V||F|$ as we install a function at each iteration. The complexity of the function $NC(l(d),s(d),\Pi)$ is of the order $O(l(d)^2 s(d))$. It gives us a complexity of $O(l_{max}^2 s_{max}|V|^2|F|^2|D|)$, when a naive algorithm would be of order $O(l_{max}^{s_{max}}|V|^2|F|^2|D|)$, as it would generate all proper cuts.

In the following page, we describe the SFCFastGreedy algorithm, and the recusive method as well.

Algorithm 1 SFCFastGreedy Algorithm

```
Require: Set of demands $D$.
Ensure: Placement $\Pi$
  for each $d \in D$ do
    $not\_hit[d] \leftarrow \binom{l(d)+s(d)-1}{s(d)-1}$
  end for
  $\Pi \leftarrow \emptyset$
  repeat
    $min\_cost \leftarrow +\infty$
    $best\_sol \leftarrow null$
    $best\_not\_hit \leftarrow null$
    for each $(u, f) \in V \times F$ do
      $newly\_hit \leftarrow 0$
      $\Pi' \leftarrow \Pi \cup \{(u, f)\}$
      for each $d \in D$ do
        $T \leftarrow new\ l(d) \times s(d)\ matrix\ of\ null$
        for $1 \leq i \leq l(d)$ do
          $T[i, 1] \leftarrow 1_{i^*(i,1)}$
        end for
        $new\_not\_hit[d] \leftarrow NC(l(d), s(d), \Pi', T)$
        $newly\_hit+ = not\_hit[d] - new\_not\_hit[d]$
      end for
      $cost \leftarrow \frac{cost(u,f)}{newly\_hit}$
      if $cost < min\_cost$ then
        $min\_cost \leftarrow cost$
        $best\_sol \leftarrow (u, f)$
        $best\_not\_hit \leftarrow new\_not\_hit$
      end if
    end for
    $\Pi \leftarrow \Pi \cup \{best\_sol\}$
    $not\_hit \leftarrow best\_not\_hit$
  until $not\_hit[d] = 0\ \forall\ d \in D$
```

```
function $NC(row\ r,\ column\ c)$
    if $T[r,c] \neq null$ then return $T[r,c]$
        $result \leftarrow 0$
        if $i^*(r,c) = 0$ then
            $result \leftarrow result + 1$
        end if
        for $0 \leq j \leq n - i^*(r,c)$ do
            $result += NC(n-j, c-1)$
        end for
        $T[r,c] \leftarrow result$
        return result
```

3.2 Joint Placement and Allocation of VNFs with Budget and Capacity Constraints

Existing SFC placement solutions mainly aim to minimize deployment cost and improve network performance. SFC Placement policies involve VNF placement strategies, which model the optimization problem with integer linear programs. Usually when VNF placement comes into consideration, SFC paths are known as part of the input.
However, VNF placement asks for an optimal investment to deploy VNFs at certain network nodes (called VNF-nodes), which has to account for practical constraints such as the deployment budget and the VNF-node capacity. To that end, it is important to design a joint VNF-nodes placement and capacity allocation algorithm that can maximize the total amount of network demands that are fully processed by the VNF-nodes while respecting such practical constraints. In contrast to most prior work that often neglects either the budget constraint or the capacity constraint, the work in [12] (Gamal Sallam and Bo Ji, IEEE INFOCOM 2019)explicitly considers both of them. Accounting for these constraints introduces several new challenges. Specifically, the authors study a problem that is not only NP-hard but also *non-submodular*. To address these challenges, they introduce a novel *relaxation* method such that the objective function of the relaxed placement subproblem becomes submodular. Leveraging this useful submodular property, they propose two approximation algorithms that achieve an approximation ratio of $\frac{1}{2}(1-1/e)$ and $\frac{1}{3}(1-1/e)$ for the original non-relaxed problem, respectively.
In the context of [12] (Gamal Sallam and Bo Ji, IEEE INFOCOM 2019), a network demand consists of a traffic load that must be fully processed at one VNF-node so that the potential benefits introduced by NFV can be harnessed. Due to the budget limit, only a subset of nodes can be selected for deploying/placing VNFs. Moreover, VNF instances typically have a limited capacity, which is shared for processing multiple passing flows. Therefore, given a deployment budget and capacity limit, it is of critical importance to choose a best subset of nodes to become VNF-nodes and to determine the optimal capacity allocation so as to maximize the amount of network traffic fully passing through them.
The authors explicitly consider both budget and capacity constraints and formulate a joint problem of VNF-nodes placement and capacity allocation(VPCA). The VPCA problem has two main components: VNF-nodes placement and VNF-nodes capacity allocation, which

are tightly coupled with each other. That is, deciding where to place the VNF-nodes depends on how the capacity of the VNF-nodes will be allocated; determining an optimal capacity allocation apparently depends on where the VNF-nodes are placed. The challenges posed by this problem are two folds. First, the placement and capacity allocation subproblems are both NP-hard. Second, the placement subproblem is *non-submodular*. This is in stark contrast to the studied problem in [13] (K. Poularakis et al., IEEE INFOCOM, 2017)that does not include the capacity constraint, which has been shown to be submodular and can be approximately solved using efficient greedy algorithms.
Retrieving useful information from the work in [12](Gamal Sallam and Bo Ji, IEEE INFOCOM 2019), we present a VNFs placement problem that can be solved when demands of single VNFs must be placed in substrate graphs, where nodes are knapsacks of specified capacities and all demands are characterized by foreknown paths. A budget (usually monetary) limits the number of VNF-nodes that can be selected for the installation of VNFs.

3.2.1 System Model And Problem Formulation

Let us consider a network graph $G = (V, E)$, where V is the set of nodes, and E is the set of edges connecting nodes in G. We have a set of demands D. A demand $d \in D$ is modeled by a couple composed of a virtual network function $f_d = f$ that has to service flow w_f and a predefined (e.g. a shortest) path whose set of nodes is denoted by V_f. Thus, we have a set of flows F, with $F = |F|$. The traffic of flow w_f will be sent along the predetermined path of the demand.
We use F_U to denote the set of all functions whose path has one or more nodes in a given set U, i.e., $F_U = \{f \in F | V_f \cap U \neq \emptyset\}$. When a node is able to support some VNFs, we call it a VNF node. Since ISPs have a limited budget to deploy VNFs in their networks, they can only choose a subset of nodes $U \subseteq V$ to become VNF nodes. We request that the traffic rate w_f of each function flow cannot be split and must be processed at a single VNF-node.
In the context of the relaxed, linear programming version of the capacity allocation subproblem, we use w_f^u to denote the portion of flow w_f that is assigned to VNF-node u and use $W = (w_f^u) \in R^{F \times V}$ to denote the assignment matrix of the relaxed problem.
With $A = A_U$ we denote the set of pairs (u, f) that includes the **assignment** of virtual function f to node u. The assignment set A_U expresses the allocation of virtual functions to the selected VNF-nodes , that have been elected according to the budget constraint.
The benefits of processed traffic can be harnessed from fully processed flows, i.e., flows that have all of their traffic processed at some VNF-node. The total processed traffic can be expressed as follows:

$$J_1(U, A) = \sum_{f \in F} \sum_{u \in V_f \cap U} w_f \chi_A(u, f) \tag{3.2}$$

where $\chi_A(u, f)$ is the indicator function, that is equal to 1 only if function f has been assigned to VNF-node u according to the assignment A. Note that each VNF-node u has a limited processing capacity, denoted by c_u. Hence, the total traffic rate assigned to a node should satisfy the following capacity constraint:

$$\sum_{f \in F} w_f \chi_A(v, f) \le c_v, \forall v \in U \tag{3.3}$$

We assume that the largest traffic flow of any function is no larger than the smallest processing capacity of any node. Also, we consider a limited budget, denoted by B, and require that the total cost of introducing VNF-nodes do not exceed B. We use b_u to denote the cost of making node u a VNF-node. Hence, the total cost of VNF-nodes should satisfy the following budget constraint:

$$\sum_{f \in F} \sum_{u \in V_f \cap U} \chi_A(u, f) b_u \le B \tag{3.4}$$

The above budget constraint limits the number of nodes that can become VNF-nodes, and we may only have a subset of flows that traverse some VNF-nodes. Accounting for the above deployment budget and VNF capacity constraints, we consider a joint problem of VNF-nodes placement and capacity allocation (VPCA). The objective is to choose a best subset of nodes to become VNF-nodes and optimally allocate their capacities so as to maximize the total amount of fully processed traffic. The mathematical formulation of the VPCA problem is the following:

$$\begin{aligned} \max_{U \subseteq V, A} \quad & J_1(U, A) \\ \text{s.t.} \quad & (3.3), (3.4) \end{aligned} \tag{3.5}$$

3.2.2 Hardness of Approximation for the VPCA Problem

The VPCA problem formulated in (3.5) is characterized by some special challenges. To highlght this, we first decompose the VPCA problem into two subproblems: 1) placement: how to select a subset of nodes to become VNF-nodes and 2) allocation: for a given set of VNF-nodes, how to divide their capacity for processing a subset of flows. It is proved that both these subproblems are NP-hard and that the placement subproblem is non-submodular. This is very different from similar problems neglecting the capacity constraint (3.3) [12](Gamal Sallam and Bo Ji, IEEE INFOCOM 2019), which have been shown to be submodular and can be approximately solved.
First, we present the formulations of the two subproblems. We start with the allocation subproblem because it will be used in the placement subproblem. For a given set of VNF-nodes $U \subseteq V$, let $J_2^U(A)$ denote the total amount of fully processed traffic under VNF assignment A. Note that $J_2^U(A)$ has the same expression as that of $J_1(U, A)$ in Eq. (3.2). The superscript U of $J_2^U(A)$ is to indicate that it is associated with a given set U of VNF-nodes. Then, the capacity allocation subproblem for a given set U of VNF-nodes can be formulated as:

$$\max_{A:(3.3) holds} \quad J_2^U(A) \tag{3.6}$$

Let $J_3(U) = \max_{A:(3.3) holds} J_2^U(A)$ denote the optimal value of problem (3.6) for a given set of VNF-nodes U. Then, the placement subproblem can be formulated as:

$$\begin{aligned} \max_{U \subseteq V} \quad & J_3(U) \\ \text{s.t.} \quad & (3.4) \end{aligned} \tag{3.7}$$

Note that in order to solve problem (3.7), we need to solve problem (3.6) to find the optimal A for a given set of VNF-nodes U. In the following theorem, we will show that both subproblems (3.6) and (3.7) are NP-hard.

Theorem 1. The capacity allocation subproblem (3.6) and the placement subproblem (3.7) are both NP-hard [12](Gamal Sallam and Bo Ji, IEEE INFOCOM 2019).

Proof. We start by proving that the allocation subproblem (3.6) is NP-hard. The proof is by a reduction from a special case of the Single Knapsack (SK) problem, where for each item the profit and the weight are identical. In the SK problem, we have a knapsack k and a set of items I. The knapsack has a capacity W, and each item $i \in I$ has a weight of w_i, which is the same as the profit. The objective is to find a subset of items $I' \subseteq I$ that has the maximum total profit and can be packed in the knapsack without exceeding its capacity. Given an arbitrary instance $A = (k, I)$ of the SK problem, we construct an instance $D = (V, F)$ of the allocation problem (3.6). The set V has only one node v_1 with a capacity that is equal to the capacity of the knapsack k. Each virtual function $f \in F$, having weight w_f corresponds to an item in $i \in I$. Node v_1 is the only VNF-node, and all the flows traverse node v_1. If we can solve the instance D of problem (3.6), the subset of functions/flows assigned to node v_1, which has the maximum total traffic rate, can be mapped to the corresponding items and solve the instance A of SK problem. Similarly, a solution for instance A of the SK problem can be mapped to a solution for instance D by simply mapping the selected items I' to the corresponding flows that solve the instance D of problem (3.6).

Next, we prove the NP-hardness of the placement subproblem (3.7). The proof is by a reduction from the Budgeted Maximum Coverage (BMC) problem. In the BMC problem, we have a set of points M and a set of candidate locations S. Each point $m \in M$ has a weight of w_m. Each location $s \in S$ has a cost of b_s and covers a subset of points $M_s \subseteq M$. The objective is to select a subset of locations $S' \subseteq S$ such that the total weight of the points covered by at least one location in S' is maximized while the total cost of the selected locations does not exceed a given budget B. Given an arbitrary instance $A = (M, S, B)$ of BMC, we will construct an instance $D = (F, V, B)$ of problem (3.6) as follows. Each function $f \in F$ corresponds to a point $m \in M$; the rate of a flow is equal to the weight of the corresponding point. Each node $u \in V$ corresponds to a location $s \in S$; the cost of a node is equal to that of the corresponding location. The path of a flow consists of the nodes corresponding to the locations that cover the point corresponding to this flow. The deployment budget of the instance D is equal to the budget of the instance A. All the nodes have an infinite capacity. We will show that a solution for the instance D exists if and only if a solution for the instance A exists. If we can solve the instance A of BMC, the subset of locations $S' \subseteq S$ that solves A of BMC can be mapped to the corresponding nodes in V to become VNF nodes and solves the instance D of problem (3.6). Similarly, if we solve the instance D, then the obtained set of VNF nodes can be mapped to the corresponding subset of locations that solve the instance A of BMC.

3.2.3 Proposed Algorithms

Let's assume the case where VNF-nodes have uniform costs, i.e., $b_v = b$ for all $v \in V$. Then, the budget constraint can be expressed as a cardinality constraint, i.e., $|U| \leq k$, where $k = \lfloor B/b \rfloor$. In this case, we can use a 2-steps algorithm to approximately solve both placement and capacity allocation problems.
We first solve the VNF-nodes placement problem: we start with an empty set of VNF-nodes U; in each iteration, we add a node that has the maximum marginal contribution to U, i.e., a node that leads to the largest increase in the value of the objective function (We choose the best k nodes that are eligible to host most VNFs). We repeat the above procedure until k VNF-nodes have been selected. This solution has been shown to achieve an approximation ratio of $(1 - 1/e)$ [17](S. Khuller, et al., 1999).

Algorithm 2 Greedy Choosing of VNF Nodes

```
Require: Set of nodes $V$, Set of demands $D$, cardinality $k$
Ensure: Set of VNF Nodes $U$, Set of supported demands $D_U$
  $U \leftarrow \emptyset$
  $D_U \leftarrow \emptyset$
  $T \leftarrow D$
  while $|U| \leq k$ do
      $u \leftarrow \arg\max_{v \in V}(|\{d \in T | v \in d\}|)$
      $D_u \leftarrow \{d \in T | u \in d\}$
      $U \leftarrow U \cup u$
      $D_U \leftarrow D_U \cup D_u$
      $T \leftarrow T \setminus D_u$
  end while
```

In the second step, we make the capacity allocation decision according to the Multiple Knapsack with Assignment Restrictions (MKAR) Approximation Algorithm.

3.2.4 MKAR LP Rounding: A 1/2 Approximation Algorithm

In MKAR, we are given a set of items $N = \{1, ..., n\}$ and a set of knapsacks $M = \{1, ..., m\}$. Each item $j \in N$ has a positive real weight w_j and each knapsack $i \in M$ has a positive real capacity c_i associated with it. In addition, for each item $j \in N$ a set $A_j \subseteq M$ of knapsacks that can hold item j is specified. Let $B_i \subseteq N$ be the set of items that can be assigned to knapsack i. We say item j is admissible to knapsack i, if $j \in B_i$.

In a feasible assignment of items to knapsacks, for each knapsack $i \in M$, we need to choose a subset S_i of items in N to be assigned to knapsack i, such that:

1. All S_i's are disjoint. (Each item is assigned to at most one knapsack.)
2. Each S_i is a subset of B_i , for $i = 1, ..., m$. (Assignment restrictions are satisfied.)
3. $\sum_{j \in S_i} w_j \leq c_i$, for $i = 1, ..., m$. (Total weight of items assigned to a knapsack does not exceed the capacity of the knapsack.)

Without loss of generality we assume that $w_j \leq c_i \; \forall j \in B_i$, otherwise j can be removed from B_i . The problem becomes trivial if all A_j's are disjoint, or if $\sum_{j \in S_i} w_j \leq c_i, \forall i \in M$. In the case where all B_i's are disjoint, the problem decomposes into m single 0-1 knapsack problems.
The assignment restrictions can be represented by a bipartite graph, where the two disjoint node sets of the bigraph correspond to the sets N and M. Let $G = (V, E)$ be the corresponding bipartite graph with $V = N \cup M$. Then, there exist an edge $(j, i) \in E$ between nodes j and i if and only if $j \in B_i$.
The IP formulation of the MKAR problem is as follows:

$$\text{maximize} \quad \sum_{i \in M} \sum_{j \in B_i} w_j x_{ij}$$

$$\text{subject to} \quad \sum_{j \in B_i} w_j x_{ij} \leq c_i, i \in M$$

$$\sum_{i \in A_j} x_{ij} \leq 1, j \in N$$

$$x_{ij} \in \{0, 1\}, i \in M, j \in N$$

where the 0-1 variable x_{ij} denotes whether item j is assigned to knapsack i. Let us denote the LP relaxation by LP-MKAR, which is obtained by replacing the constraints $x_{ij} \in \{0, 1\}$ with the constraints $0 \leq x_{ij} \leq 1$.
LP-MKAR can be solved by a maximum flow algorithm on a directed graph constructed from the bigraph as follows. Each edge (j, i) of G is directed from the item node j to the knapsack node i and is assigned capacity w_j . A source node s is connected to each item node j via an arc (s, j) with capacity w_j . In addition, a sink node t is connected to each knapsack node i via an arc (i, t) with capacity c_i . Then, the maximum flow from s to t equals the LP relaxation value and the amount of flow on arc (j, i) divided by w_j gives the value of x_{ij}. Thus, if flow on (j, i) equals w_j , i.e., $x_{ij} = 1$, then item j is assigned to knapsack i. If $0 < x_{ij} < 1$, the variable is said to be fractional (in the corresponding solution).
Let x be an optimal basic feasible solution to LP-MKAR and let f denote the vector of fractional variables of x. We denote the subgraph of the bigraph representation G induced by the edges in f by R_G and call it the *residual* graph.

Lemma 1. The residual graph, R_G, of the LP relaxation is a forest [15](M. Dawande et al., 2000).
We can consider each connected component of R_G separately. Let OPT denote the optimum value of LP-MKAR and let $AW(K_i)$ denote the total *Assigned Weight (AW)* to knapsack K_i in x. In R_G, if a knapsack has degree 1, then we refer to that knapsack as a *singleton* knapsack.

Lemma 2. If R_G has a singleton knapsack node, say K_1, then there is a rounding step for knapsack K_1 such that $R_1 \geq OPT - \frac{1}{2}AW(K_1)$, where R_1 denotes the objective function value of the solution obtained after the rounding step [15](M. Dawande et al., 2000).
Proof of Lemma 2. Let S denote the set of items which have been assigned (integrally) to knapsack K_1 by the solution x. Note that S is nonempty, because the optimal solution x of the LP-MKAR would not choose a knapsack K_1 to assign only one item fractionally to it. Fractional assignment for an item can happen only to satisfy maximum assignment provided that all chosen knapsacks have been already paired with some fully assigned items. Let $W(S)$ denote the total weight of items in S. In addition, let I_1 be the item which is fractionally assigned to knapsack K_1 with value f_1. If $W(S) \geq w_1 f_1$, then we round f_1 down to zero. Now K_1 has no fractional variable incident on it. Let us denote the new solution by x_1. Then, x_1 has objective value $R_1 = OPT - w_1 f_1$. Since $AW(K_1) = w_1 f_1 + W(S) \geq 2w_1 f_1$, it is true that $R_1 \geq OPT - \frac{1}{2}AW(K_1)$. On the other hand, if $w_1 f_1 \geq W(S)$, then we unassign all items in S and round f_1 to 1. We also round all other fractional variables incident on I_1 to 0. Then, $R_1 \geq OPT - W(S) \geq OPT - \frac{1}{2}AW(K_1)$.

Lemma 3. Using Lemma2, Algorithm 3 (as described above), has an approximation ratio of $1/2$ [15](M. Dawande et al., 2000).

It follows that VPCA problem in (3.5) can be solved with an approximation of $1/2(1 - 1/e)$.

3.3 The Set-Union Knapsack problem (SUKP)

Before we proceed to the next chapter, where we propose a new problem formulation, we would like to provide some information for the Set-Union Knapsack problem (SUKP). The reason is that, we use SUKP as a subproblem in the proposed solution of our definition.

The Set-Union Knapsack problem (SUKP) is a generalization of the 0-1 knapsack problem. The SUKP comprises of a set of elements $U = \{1, ..., n\}$ and a set of items $S = \{1, ..., m\}$. Each item, $i = 1, ..., m$, corresponds to a subset of elements, denoted by S_i, with a nonnegative profit given by $p : S \rightarrow R_+$ and each element has a nonnegative weight given by $w : U \rightarrow R_+$. For a subset $A \subseteq S$, we define the weighted union of set A as $W(A) = \sum_{e \in \cup_{i \in A} S_i} w_e$ and $P(A) = \sum_{i \in A} p_i$. We want to find a subset of items $S^* \subseteq S$

Algorithm 3 MKAR Based, SC-Single Function Allocation Algorithm

```
Require: Set of VNF-nodes $U$, Set of functions $F_U$ , function sizes $w_f$, and VNF-node
    capacities $c_u$.
Ensure: Allocation of Functions to nodes $A_U$
  $A_U \leftarrow \emptyset$
  /* Phase I - Solve Fractional LP-MKAR */
  Obtain a basic optimal solution $W_U$;
  $x_{uf} \leftarrow w_f^u / w_f$ for all $w_f^u \in W_U$;
  /* Assign each function with $x_{uf} = 1$ to VNF-node u */;
  if $x_{uf} = 1$ then
      $A_U \leftarrow A_U \cup (u, f)$
  end if
  /* Phase II - Modify residual graph for the unassigned functions with positive $w_f^u$ */
  $G^{'} \leftarrow R_G(W_U)$; /* $R_G$ is the residual graph of fractional solution */
  while $\exists$ singleton VNF-node $u_1$ with fractional assignment $f_1$ under edge $e_1$ in $G^{'}$ do
      $S \leftarrow \{f \in F | (u_1, f) \in A_U\}$ /* $S \neq \emptyset$ */
      if $W(S) \geq w_1 f_1$ then
          /* Removing edge $e_1$ from residual graph makes $u_1$ a non-singleton node */
          $G^{'} \leftarrow G^{'} \setminus e_1$; /*Remove single edge of fractional weight */
      else
          $A_U \leftarrow A_U \setminus \{(u_1, f) \in A_U \, | f \in F\}$
          /* Rounding $f_1$ to 1, makes $u_1$ a non-singleton node */
          $A_U \leftarrow A_U \cup (u_1, f_1)$
          while $\exists u \in V$: $0 < w_{f_1}^u < 1$ do $w_{f_1}^u = 0$
          end while
      end if
  end while
```

such that $P(S^*)$ is maximized and $W(S^*) \leq B$, where B is a given budget. Goldschmidt et al. [16] studied the problem and presented a dynamic programming algorithm running in exponential time to solve the problem exactly. The SUKP remains NP-hard, even in very restricted cases.

A closely related problem is the densest k-subhypergraph problem [19](V.V. Vazirani et al., 2006), in which we are given a hypergraph $H(V, E)$ and we have to determine a set of k nodes such that the subhypergraph induced by this set has a maximum number of hyperedges. SUKP reduces to the densest k-subhypergraph problem (DkH), when we have unit weights and unit profits, with the elements and items corresponding to the nodes and hyperedges respectively and the budget being k. It has been shown [19](V.V. Vazirani et al., 2006)that the densest k-subhypergraph problem cannot be approximated within the factor of $2^{(logn)^{\delta}}$, for some $\delta > 0$, unless $3SAT \in DTIME(2^{n^{\frac{3}{4}+\epsilon}})$. For the special case, where we have the item size equal to exactly 2, we have the densest k-subgraph (DkS) problem. The best known algorithm provides an approximation factor of $O(min\{n^{\delta}, n/k\})$, for any $\delta \leq 1/3$ [20](U. Feige et al., 2001).

3.3.1 Approximation for a special case of the Set-Union Knapsack problem

Based on the greedy algorithm for the budgeted maximum coverage problem developed by Khuller,Moss and Naor in [17], the author in [18] presents a greedy algorithm for the SUKP with the additional restriction that the number of items in which an element is present is bounded by a constant d. It is proved in [18] that the algorithm provides a $(1 - e^{-\frac{1}{d}})$ approximation. This factor naturally extends to densest k-subgraph problem where the input graph has a bounded degree.

The algorithm in depicted below, as Algorithm-5. There are some notations needed though. We define d_e as the frequency of an element e, i.e., the number of items in which element e is present. So, we have $\max_{e \in U} d_e \leq d$. For an item i, we denote the profit of item i by p_i and define $W_i' = \sum_{e \in S_i} \frac{w_e}{d_e}$, where w_e is the weight of element e. We consider all possible subsets of items of cardinality 2 or less, whose weighted union is within the budget B. We augment each of these subsets with items (not in the subset) one by one in the decreasing order of the ratio $\frac{p_i}{W_i'}$, if its inclusion does not violate the budget B. We then choose of the best of these augmented sets as A. The author in [18] pointed out that the items could be considered in the increasing order of the ratio of sum of weights of elements in the item that are yet to be picked to its profit and this will ensure the same guarantee on the approximation factor, but it is easier to follow the analysis of Khuller et al. with the one presented in [18]. For the greedy augmentation part we use the Budgeted Maximum Coverage Greedy algorithm as a subroutine (Algorithm-4).

Algorithm 4 GREEDY(G,U,B)

```
Require: Set of items $U$, Budget $B$
Ensure: Set of items with highest profit to weight ratio $G$
  while $U \neq \emptyset$ do
      $i \leftarrow \arg\max_{i \in U}(\frac{p_i}{W_i'})$
      if $W(G \cup \{i\}) \leq B$ then
          $G \leftarrow G \cup \{i\}$
      end if
      $U \leftarrow U \setminus \{i\}$
  end while
```

Algorithm 5 A-SUKP(G,B)

```
Require: Set of subsets $G$, Budget $B$
Ensure: Set of subsets of highest profit $A$
  $A \leftarrow \emptyset$
  for each $G \subset S$ such that $|G| \leq 2$, $W(G) \leq B$ do
      $G \leftarrow GREEDY(G, S \setminus G)$

      $A \leftarrow \arg\max(P(G), P(A))$
  end for
```

We want to point out that

$$\lim_{d \to \infty} \left| (1 - e^{-\frac{1}{d}}) - \frac{1}{d} \right| = 0$$

This means that the true approximation factor is actually the constant $1/d$.

4. CONTRIBUTION

When Telco Cloud Operators have to deal with the placement or routing of Service Function Chains in substrate physical data networks, they need to take into account a set of restrictions and requirements.
First of all, they usually have a limit on the CPU resources, that are available for the installation of VNFs in the given VNF nodes. The available CPU may be a global quantity of virtual cores in a distributed farm of data centers, or it may refer to physical servers of defined capacities, where VNFs can be assigned. Since each VNF incurs a weight in the installation, equal to its CPU consumption, and each SFC is a set of VNFs, we ask for a policy that chooses a subset of SFCs which fits into the available CPU budget. In real scenarios, the operators have the freedom to distribute the CPU budget in the networks' VNF nodes as they desire, however, this decision should be done with some optimality criterion in mind. Usually, SFCs are characterized by two metrics, these are the flow amount processed by SFC VNFs and the latency of the path where VNFs are placed and route their packets. Ideally, network designers want to place SFCs that provide the maximum flow under processing, with the minimum latency achieved.
Another important requirement is that each VNF node can support specific VNFs. This means that some VNFs can be enabled in specific nodes, while others on different nodes of the network.

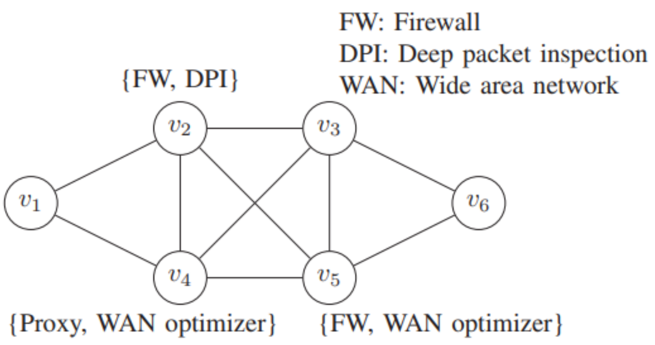

Figure 4.1: A network with network functions at different locations

An example is provided in Figure 4.1 in which different network functions are supported at different locations. Assume that we have a flow from v_1 to v_6, with the following SFC constraint: $(v_1, Firewall\ (FW),\ wide\, area\, network(WAN)\ optimizer,\ v_6)$. Different paths that satisfy the SFC constraint are available for this flow such as $(v_1, v_2, v_4, v_3, v_6)$, or (v_1, v_4, v_5, v_6). Which of these paths to choose depends on the load at each instance and the total congestion/latency along each path.
In this chapter we formulate a problem of joint SFC selection and path routing, with given CPU budget. We formulate an optimization objective that asks to maximize the total routed SFC flows under minimum path latency. We actually seek to route those SFCs that provide the maximum total processed data rate. We propose an algorithm for this problem, and try to sketch an approximation ratio for it, thus suggesting a problem (and a possi-

ble solution) for an approximate solution of SFC placement/routing under VNF chaining constraints.

4.1 System Model and Problem Formulation

Let us consider a graph $G = (V, E, T)$ (V is the set of nodes, E is the set of edges and T is the set of time delays upon the edges), a set of virtual functions F, and a collection D of demands. Each node $u \in V$ of the graph can support a subset $F_u \subset F$ of network functions. Each demand $d \in D$ is associated with an ordered sequence of functions $sfc(d) = (f_1, f_2, ..., f_r) \in F^*$ and a volume of packets $vol(d) \in R_+$. A (setup) weight $w_f : F \rightarrow R_+$, defines the CPU consumption of function $f \in F_u$, when this function is activated in node $u \in V$. When a function is activated in some VNF-node, then the function's CPU cost is paid once and only. Different SFCs can use the same function to route their packets through that function, if it is decided so. A CPU budget C limits the total number of service function chains that can be selected in the network.

Definition of function placement: For a demand $d \in D$ we denote with P_d the set of all directed paths in G that respect the SFC constraint represented as: $(u_s, f_1, ..., f_r, u_d)$, where u_s and u_d are the source and destination, respectively, and $(f_1, ..., f_r)$ denotes a sequence of network functions by which all the packets of the flow need to be processed before reaching u_d. A path $p \in P_d$ has a trip time cost $trip\_time(p) \in R_+$ which is the sum of the delays of the edges of the path. A feasible *function placement* is a path $p \in P_d$ whose nodes that support each function $\{f_i\}_{i=1}^{i=r}$ are used to host the functions of the chain $sfc(d) = (f_1, f_2, ..., f_r)$.

Objective: maximize

$$\sum_{d \in D} \sum_{p \in P_d} \frac{vol(d)}{trip\_time(p)} x_{pd} \tag{4.1}$$

Where:

$$x_{pd} = \begin{cases} 1, & \text{p is chosen for SFC placement for demand d} \in D \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

With:

$$\sum_{p \in P_d} x_{pd} \leq 1, \, \forall d \in D \tag{4.3}$$

And:

$$\sum_{d \in D} \sum_{p \in P_d} \sum_{f \in sfc(d)} w_f x_{pd} \leq C \tag{4.4}$$

The maximization of the objective function (4.1) asks to find and route to the network these service functions chains that lead to the maximum routed data rate in total (i.e. Mbits/sec). Constraints (4.2) and (4.3) state that only one path can be selected to route the packets of one service chain, whereas constraint (4.4) denotes that the total weight of the functions of the chains cannot be larger that the available CPU budget C. This is a capacitated routing problem, that can be found in Cloud environments, where administrators must decide which chains to route in the network and under which paths. We call this the Maximum Data Rate SFC Routing under Global Capacity Constraint Problem (MDR-SFC-GCC).

4.2 Routing in the scene

In order to maximize the objective function of the problem, we need to evaluate the coefficients of the form in (4.1). This means we need to estimate the $trip\_time(p)$ for the paths of the demands. We will use a **greedy strategy that finds the timely shortest path**, which in the same time it respects the sequence of the functions $(u_s, f_1, f_2, ..., f_r, u_d)$ that obliges the packets of a demand d to pass through its predefined order of virtual functions. The evaluation of the timely shortest path results in having the largest possible data rate profits for each demand, independently of its weight. So, in a first step we calculate the ”VNF-functions constrained” timely shortest path for each demand of the input set.
Using the configuration graph for each demand we can find the best $trip\_time(p)$ of the timely shortest paths $p \in P_d, \forall d \in D$ in polynomial time. This path will be the route of a demand if the demand is finally selected. The configuration graph is a layered graph (Figure 4.2), where the nodes of each layer are the nodes of the original graph that are able to host the function of the respective layer of the chain. The cost of each edge is the cost of the timely shortest path between the respective VNF-nodes of the original graph. In this context, the path with the best $trip\_time(p)$ is the shortest path in the configuration graph, that can be found using a shortest path finding algorithm like Djkstra’s algorithm.

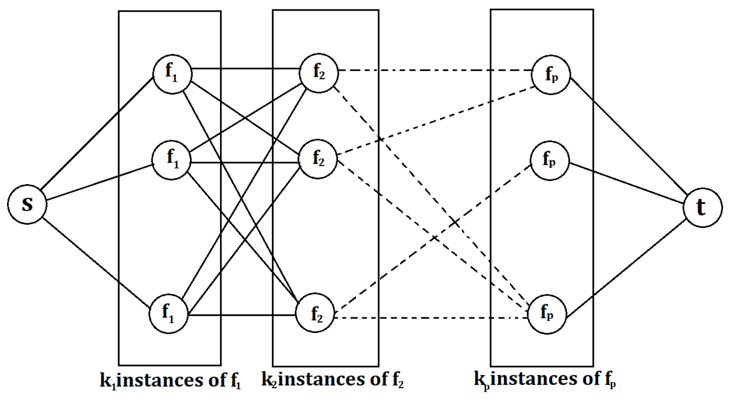

Figure 4.2: Configuration graph - general case

Figure 4.3 shows a configuration graph (CG) for a given network and service chain. As easily observed, we have two nodes of FW and WAN functions in each layer of the CG, since two nodes support the specific functions in the initial graph.

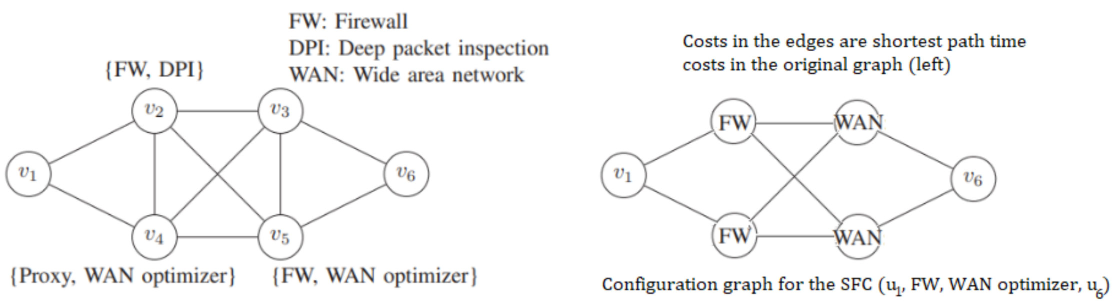
FW: Firewall
DPI: Deep packet inspection
WAN: Wide area network
{FW, DPI}
v_1
v_2
v_3
v_4
v_5
v_6
{Proxy, WAN optimizer}
{FW, WAN optimizer}
Costs in the edges are shortest path time costs in the original graph (left)
FW
WAN
FW
WAN
v_1
v_6
Configuration graph for the SFC (u_1, FW, WAN optimizer, u_6)

Figure 4.3: Configuration graph for SFC path finding

After evaluating the $trip\_time(p)$ denominators of the profits in the form (4.1), we can proceed to the selection of the best service function chains against the given CPU budget. We have to investigate a capacitated set union problem with known profits, but without knowing a priori whether the sets share elements or not.

4.3 Solving MDR-SFC-GCC Problem

After finding the optimal path for each demand, we put in a ground set all the functions of the paths of the service chains. There could be functions of the same operation type residing in different nodes of the network. Routing phase can give paths sharing the same function/node or not. As a result, in order to maximize the cumulative data rate of the satisfied service chains, we check the following conditions:

- If the routing phase resulted in paths that do not share functions, then we have to solve a knapsack problem for which an FPTAS scheme exists. In this case we collect all the functions of a chain's path as one item, having weight equal to the summary of the weights of the functions.
- If the routing phase resulted in paths that share functions/nodes, then we have to solve a Set-Union knapsack problem.

So, if we assume that there is a constant bound q on the shared VNFs among the service chains, which can be a real condition in cloud environments, we propose an algorithm with guaranteed ratio, this of $(1 - e^{-\frac{1}{q}})$. We do this by using the Algorithm-5 presented in subsection 3.3.1 (special instance for SUKP problem). In realistic conditions, this can happen provided that a given number of the VLAN ports in each VNF is part of the input, which makes the SFC paths sharing a VNF also bounded. In this case the problem becomes harder, as we need to solve a separate (multidimensional) knapsack problem in a first place which needs further investigation and analysis.

Algorithm 6 MDR-SFC-GCC-Algorithm(D,C)

```
Require: Set of demands $D$, CPU Budget $C$
Ensure: Set of demands with highest data rate $\tilde{D}$
  $paths \leftarrow new\ vector\ of\ length\ |D|\ of\ null$
  for each $d \in D$ do
      $paths[d] \leftarrow Configuration\_Graph\_Shortest\_Path(d)$
      $profit[d] \leftarrow \frac{vol(d)}{cost(paths[d])}$
  end for
  $items \leftarrow paths, items.profit \leftarrow profit[d], items.weight \leftarrow weight[d]$
  if $\forall i \neq j,\ paths[i].nodes \cap paths[j].nodes = \emptyset$ then
      $\tilde{D} \leftarrow Knapsack\_FPTAS(items, C)$
  else
      $\tilde{D} \leftarrow A - SUKP(items, C)$
  end if
```

5. OPEN PROBLEMS

In this chapter we talk about open problems around SFC/VNF placement. At a first place, we give the generalization of a problem we saw in the previous chapter.

5.1 Joint Placement and Allocation of SFCs with Budget and Capacity Constraints

In paragraph (3.2) we investigated the problem of Joint Placement and Allocation of VNFs with Budget and Capacity Constraints. A natural generalization of this problem, comes to light when we want to place SFCs instead of single VNFs. The problem is very hard, considering that we need to take into consideration the precedence constraints between the VNFs of the service chains.
We give the following definition of the problem:

Input: A digraph $G = (V, E)$, a set of functions F, and a collection D of demands. Each demand $d \in D$ is associated with a path $path(d) \in V^*$, a sequence of functions $sfc(d) \in F^*$ and a flow $p(d) \in R_+$. A (setup) weight $w : F \to w_f$, defines the weight (i.e. CPU/Memory) of function f in any node. Each node $v \in V$ has a capacity c_v. A (usage) cost $b : V \to b_v$, defines the cost (i.e. monetary) of using node v for setting up at least one VNF.

Definition of function placement: A *service functions chain placement* is a subset $\Pi \subset V \times F$ of function locations. We say that a demand $d \in D$ associated with a path $path(d) = u_1, u_2, ..., u_{l(d)}$ and a chain $sfc(d) = r_1, r_2, ..., r_{s(d)}$ is satisfied by Π, if there exists a sequence of indices $i_1 \leq i_2 \leq ... \leq i_{s(d)}$, such that $(v_{i_j}, r_j) \in \Pi$, for $1 \leq j \leq s(d)$.

Output: An *optimal service functions chain placement* that is a subset $\Pi \subset V \times F$ of function locations, such that, demands $d \in D$ of maximum cumulative flow are satisfied.

Objective: maximize

$$\sum_{d \in D} p(d) f(\Pi, d) \tag{5.1}$$

Where:

$$f(\Pi, d) = \begin{cases} 1, & \Pi \text{ satisfies d} \\ 0, & \text{otherwise} \end{cases} \tag{5.2}$$

under the constraints:

$$\sum_{f \in F} \chi_\Pi(v, f) w_f \leq c_v, \forall v \in V \tag{5.3}$$

$$\sum_{v \in V} \sum_{f \in F} \chi_\Pi(v, f) b_v \leq B \tag{5.4}$$

The quantity B is a budget. With $\chi_\Pi(v, f)$ we mean the indicator function showing whether policy Π places function f in node v or not. A function can be placed in at most one node, which leads to:

$$\sum_{v \in V} \chi_{\Pi}(v, f) \leq 1, \forall f \in F \tag{5.5}$$

5.2 General cases

VNF placement and SFC routing joint optimization solutions have the optimization objectives in both VNF-level (mainly consider deployment cost, resource usage, etc.) and routing-level (mainly consider link utilization, delay, etc.). Because of the conflict between these two levels, the optimization solutions need to balance the objectives of VNF-level and routing-level according to the requirements of network operators and users. Furthermore, in order to realize fast solving in large-scale network, these solutions propose different approximation algorithms to exchange the accuracy of optimization results for lower computational complexity.

The MDR-SFC-GCC problems asks for further investigation, in the direction which will expose the conditions needed for having a guaranteed approximation solution for the problem. We hope that the problem definition will trigger the reader to search for more information around similar problems, inspiring them to propose sophisticated techniques in the area.

Important existing open problems ask for the maximization of resource utilization and/or minimization of average path latency, with proved upper/lower bounds of algorithm performance. Energy consumption of VNFs is another important parameter that is taken into consideration when designing approximation algorithms for the specific domain.

A remaining unaddressed issue is considering flow rates and the accounting of practical constraints such as *soft capacities* on network functions or *hard capacities* on network nodes. An interesting future research direction may concern an investigation of the possibility of efficiently approximating these problems.

Of course online versions of the discussed problems are always in the centre of industry's interest, as in real conditions VNFs appear and die dynamically in cloud data centers.

Last but not least, we note the importance of studying certain topologies of the problems (trees, chordal graphs), which can ease the finding of approximations for some instances.

6. CONCLUSION

In this thesis we studied significant approximation algorithms for SFC/VNF placement, targeting to present techniques of guaranteed approximation solutions. We showed how instances of the SFC/VNF placement problem reduce to known problems, like the Multiple Knapsack With Assignment Restrictions Problem and the Budgeted Maximum Coverage Problem. We presented a problem definition that arises in cloud environments, where operators need to route the most profitable (in terms of data rate) service chains under a general capacity constraint. Finally. we sketched a solution for a special instance of the problem.

ABBREVIATIONS - ACRONYMS

NFV	Network Function Virtualization
VNF	Virtual Network Function
SFC	Service Functions Chain
TELCO	Telecommunications Company
MKAR	Multiple Knapsack With Assignment Restrictions Problem
SUKP	Set Union Knapsack Problem
BMC	Budgeted Maximum Coverage Problem

REFERENCES

[1] Xin Li and Chen Qian, ”A Survey of Network Function Placement”, Department of Computer Science, University of Kentucky, 13th IEEE Annual Consumer Communications Networking Conference (CCNC), 2016.

[2] Juliver Gil Herrera and Juan Felipe Botero, ”Resource Allocation in NFV: A Comprehensive Survey”, IEEE Transactions on Network and Service Management (Volume: 13, Issue: 3, September 2016).

[3] R. Guerzoni et al., ”Network functions virtualisation: an introduction, benefits, enablers, challenges and call for action”, introductory white paper. In SDN and OpenFlow World Congress, 2012.

[4] ETSI. European Telecommunications Standards Institute. Accessed on Feb. 28, 2016.

[5] A. Gemberet al., “Stratos: A network-aware orchestration layer for middleboxes in the cloud”, CoRR, vol. abs/1305.0209, 2013.

[6] P. Quinn and J. Guichard, “Service function chaining: Creating a service plane via network service headers,” Computer, vol. 47, no. 11, pp. 38–44, Nov. 2014.

[7] A. Tomassilli, F. Giroire, N. Huin, and S. Perennes, ”Provably Efficient Algorithms for Placement of Service Function Chains with Ordering Constraints”, IEEE INFOCOM 2018 - IEEE Conference on Computer Communications.

[8] V. Chvatal, “A greedy heuristic for the set-covering problem”, Mathematics of operations research, vol. 4, no. 3, pp. 233–235, 1979.

[9] N. Alon, D. Moshkovitz, and S. Safra, “Algorithmic construction of sets for k-restrictions,” ACM Trans. Algorithms, vol. 2, no. 2, 2006.

[10] K. Menger, “Zur allgemeinen kurventheorie,” Fundamenta Mathematicae, vol. 10, no. 1, pp. 96–115, 1927.

[11] Weihan Chen, Xia Yin, Zhiliang Wang and Xingang Shi, ”Placement and Routing Optimization Problem for Service Function Chain: State of Art and Future Opportunities”, International Conference on Artificial Intelligence and Security, 2020.

[12] Gamal Sallam and Bo Ji, ”Joint Placement and Allocation of Virtual Network Functions with Budget and Capacity Constraints”, IEEE INFOCOM 2019 - IEEE Conference on Computer Communications.

[13] K. Poularakis, G. Iosifidis, G. Smaragdakis, and L. Tassiulas, “One step at a time: Optimizing sdn upgrades in isp networks,” in Proceedings of IEEE INFOCOM, 2017.

[14] G. L. Nemhauser and L. A. Wolsey, “Maximizing submodular set functions: formulations and analysis of algorithms,” in North-Holland Mathematics Studies. Elsevier, 1981, vol. 59, pp. 279–301.

[15] M. Dawande, J. Kalagnanam, P. Keskinocak, F. S. Salman, and R. Ravi, “Approximation algorithms for the multiple knapsack problem with assignment restrictions,” Journal of combinatorial optimization, vol. 4, no. 2, pp. 171–186, 2000.

[16] O. Goldschmidt, D. Nehme, G. Yu, ”Note: on the set-union knapsack problem”, Naval Res. Logist. (NRL) 41 (6) (1994) 833–842.

[17] S. Khuller, A. Moss, J. Naor, ”The budgeted maximum coverage problem”, Inform. Process. Lett. 70 (1999) 39–45.

[18] Ashwin Arulselvan ”A note on the set union knapsack problem”, Elsevier, Discrete Applied Mathematics Volume 169, 31 May 2014, Pages 214-218.

[19] M.T. Hajiaghayi, K. Jain, K. Konwar, L.C. Lau, I.I. Mӑndoiu, A. Russell, A. Shvartsman, V.V. Vazirani, ”The minimum k-colored subgraph problem in haplotyping and DNA primer selection”, in: Proc. Int. Workshop on Bioinformatics Research and Applications, IWBRA, 2006.

[20] U. Feige, G. Kortsarz and D. Peleg ”The dense k-subgraph problem”, Algorithmica, 29 (1999), p. 2001.