University of Macedonia
Department of Applied Informatics

# Orchestrating Service Function Chains over Resource-Constrained Network Function Virtualization (NFV) Infrastructures

Angelos Pentelas

Submitted in partial fulfilment of the requirements for the degree of
Doctor of Philosophy in Applied Informatics at the University of Macedonia, October 2023

# Abstract

Network Function Virtualization (NFV) has recently emerged as a prominent cloud computing paradigm to address the scalability limitations of middleboxes, such as firewalls, load-balancers, and proxies. By decoupling them from their underlying hardware and fostering their implementation in the form of virtual network functions (VNFs), middleboxes can now be deployed on commodity servers in the form of virtual machines or containers and scale on-demand in response to evolving resource requirements. In addition, multiple VNFs can be organized into an ordered sequence to form a service function chain (SFC). Admittedly, VNFs and SFCs are the two primary constructs within the NFV ecosystem.

While NFV renders middlebox deployments more elastic and consequently cost-effective, it also introduces several challenges, primarily related to the management and orchestration of VNFs and SFCs. A particularly significant challenge is the SFC embedding (SFCE) problem, which pertains to the optimal mapping of VNFs and virtual links to their physical counterparts, namely servers and links within a substrate network. The optimization of SFCE is *NP-hard*, with both exact and approximate methods proposed in the literature. However, as the NFV landscape evolves, leaning towards localized and resource-constrained compute models such as edge clouds, the relevance and scope of prevailing SFCE methods shall be revisited.

To substantiate this, we identify four critical limitations of existing SFCE methods in resource-constrained NFV. The first one stems from SFCE optimization over a single resource dimension, *i.e.,* the CPU. While in core datacenters the CPU is indeed the main resource bottleneck, this might not hold in resource-constrained networks, given the high versatility of resource demands, along with the potentially lower capacity of remaining resource types, *e.g.,* storage, memory, etc. Another limitation of prevailing SFCE has its grounds on its lack to handle imminent cross-service communications. The latter, which are particularly attractive in edge clouds, introduce additional constraints to the standard SFCE scope, and these necessitate substantial SFCE adjustments. A third limitation results from centralized reinforcement learning (RL) schemes that can be found in the respective SFCE literature. These typically make highly optimistic assumptions with regard to the observability of the underlying network, challenging their practicality in real-life deployments. To this end, we identify the need for the investigation of more appropriate RL schemes in the context of SFCE and NFV orchestrators in general. Last, not much light has been shed on SFC pre-processing methods. As such, SFCs are mostly perceived as invariable constructs, which contradicts their very virtual structure. In extension, SFCE solvers might struggle to obtain solutions on par with the optimization objective during unfavourable network states (*e.g.,* high utilization). This raises the need for SFC pre-processing algorithms that can optimize the SFC structure according to fluctuating network conditions.

This thesis sets out to explore the aforementioned challenges. Specifically, (i) we investigate SFCE across multiple resource dimensions as a means to minimize resource wastage in resource-constrained NFV, proposing adept heuristic and exact algorithms to handle the intricacies of the problem at hand. Subsequently, (ii) we study SFCE through the lens of cross-service communications, which is a relatively new concept in the realm of NFV that advocates for cross-service interactions, thus amplifying network capabilities. Further, (iii) we examine multi-agent RL schemes within SFCE, paying particular attention to concise cross-agent communications and the interpretation of the scheme's learning capacity across various network topologies. Last, (iv) we delve into SFC graph transformations in order to adapt the SFC request according to the resource conditions of the underlying network, thus enabling SFCE solvers sustain high-quality solutions even in conditions of limited resource availability and/or high level of resource fragmentation.

# Acknowledgements

I would like to express my sincere gratitude to my supervisor, Prof. Panagiotis Papadimitriou, for his unwavering support, invaluable guidance, and continuous encouragement throughout my doctoral journey. His expertise, dedication, and mentorship have been instrumental in shaping my research and personal growth.

I am also deeply thankful to the exam committee members, Prof. Eleftherios Mamatas and Prof. Angelos Sifaleras, for their insightful feedback, constructive criticism, and valuable contributions to my work.

I am grateful for the stimulating discussions, shared insights, and constructive feedback from fellow researchers, and in particular from the Nokia Bell Labs team, namely Danny, Chia-Yu, and Koen, and the Netcloud team, namely George and Makis, which have enriched my research and helped me refine my ideas.

To my family and friends, I am profoundly grateful for their unwavering support, love, and encouragement throughout this challenging yet rewarding journey.

# Dedication

This thesis is dedicated to my family.

‘The complexity of a solution should be at most as complex as the problem space it inhabits.’

*Unknown author*

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Motivation and Objectives

The rapid growth of services on the Internet calls for advanced ways to control and manage network functions, such as firewalls, proxies, and load balancers. Traditional methods that rely heavily on hardware are struggling to handle the growing complexity and variety of these services [1]. Network Function Virtualization (NFV) has shown great potential in tackling these issues by making networks more adaptable, and thereby more efficient and cost-effective [2]. This is achieved with the introduction of NFV-enabled network constructs, such as virtual network functions (VNFs) and service function chains (SFCs). The former refer to the implementation of hardware-based network functions in the form of virtual machines and containers, whereas SFCs implement ordered sequences of VNFs, which allows them to enforce complex, end-to-end flow processing policies. Besides enterprise networks, NFV has also found traction across radio access networks (RANs) with the virtualization of, *e.g.,* basestations and gateways [3].

However, NFV presents multifaceted challenges that primarily stem from the dynamic and flexible nature of VNFs. As opposed to traditional network architectures, where network functions are tied to specific hardware, NFV decouples these functions, allowing them to be instantiated, scaled, and migrated on-demand across distributed virtual environments. This decoupling necessitates advanced orchestration mechanisms to ensure efficient allocation, scaling, and termination of VNFs while maintaining service level agreements and optimal resource utilization [4]. Alongside these, SFC embedding (SFCE) emerges as a pivotal problem within the NFV ecosystem. SFCE involves the correct and efficient placement of a set of inter-connected VNFs within a physical network (*e.g.,* one or multiple datacenters). The importance of SFCE is underscored by its impact on network performance, service delivery efficiency, and the overall quality of service. An inefficiently or incorrectly embedded SFC can result in sub-optimal network performance, increased latency, and even service disruptions.

NFV is considered a relatively new cloud computing paradigm. As such, we deem that NFV orchestration challenges, including the SFCE problem, shall be examined through the lens of an evolving NFV landscape. Admittedly, this landscape is characterized by a remarkable shift towards more decentralized and localized cloud computing models, mainly driven by the need for low latency, bandwidth conservation, and location-aware services. This shift signifies the emergence of resource-constrained environments, more specifically, edge clouds [5]. Unlike traditional centralized datacenters, edge clouds are distributed computing infrastructures situated closer to data sources (such as end-users or IoT devices). By positioning compute resources at the edge of the network, these clouds enable swift data processing, reduced data transfer costs, and improved responsiveness, which are paramount for applications like autonomous vehicles, augmented reality, and real-time analytics. However, with their resources being notably constrained compared to massive core datacenters, new challenges arise with respect to SFCE. In particular:

1. A typical assumption in core datacenters is that certain resource types apart from the CPU (*e.g.,* memory and storage) are abundant. While this holds true in some sense, it has led most existing SFCE studies optimize over a single resource type, namely the CPU, effectively claiming that the respective solvers can be easily extended to tackle multi-resource settings. We deem that it is imperative to investigate the above claim, and to measure the impact of neglecting resource dimensions during SFCE optimization in resource-constrained clouds, where resource abundance is challenged.

2. Edge cloud environments pave the way for cross-service interactions, as a means to leverage service collocation and thus amplify network capabilities. Yet, edge nodes (*e.g.,* servers) are often resource-constrained, making optimal resource allocation critical when multiple SFCs interact. Moreover, the dynamic nature of edge clouds necessitates agile scaling mechanisms, and the latter shall now account for resource scaling decisions across multiple interacting services. These features emphasize the intricacies in resource orchestration and scaling amidst interacting services, and they introduce relatively new complexities to the domain of NFV. Hence, it is important to investigate SFCE through this new perspective in order to fully harness the potential of next-generation networks while ensuring efficient resource utilization.

3. Reinforcement learning (RL) has been increasingly applied in the context of SFCE, predominantly using a centralized approach. Often, it is assumed that the centralized agent possesses a comprehensive view of the environment, which includes several interconnected edge clouds. However, this assumption seems to be in contradiction with the NFV Management and Orchestration (MANO) architecture. In the NFV MANO model (*e.g.,* [6, 7, 8]), a centralized agent is more likely to have access only to aggregate information, providing a partial, rather than a complete, insight into the true state of the system. This discrepancy underscores the need for a closer examination of how RL models align with real-world orchestration frameworks.

4. In the prevailing literature, SFC pre-processing methods have often been overlooked, resulting in an evident gap in research. Consequently, SFC requests are predominantly perceived as fixed constructs, meaning that their structure is mostly deemed immutable and invariable. This perspective severely restricts the flexibility and adaptability of SFCE solutions, hindering their optimization in alignment with the real-time state and availability of resources in the physical network. This highlights a pressing need for more adaptive methodologies that can transform SFC requests to better fit dynamic network conditions, and to augment solvers in finding high-quality solutions even at unfavourable network states.

The main goal of the thesis is to address the above challenges by investigating, developing, and validating new methods and algorithmic mechanisms revolving around SFCE. Concretely, the first goal is to develop new SFCE algorithms that take into account multiple resource dimensions. This will help to use network resources more efficiently by minimizing resource wastage. The second objective focuses on the development of SFCE methods for efficient cross-service communication. This approach is anticipated to augment the interconnected functionality of diverse, yet collocated, SFCs, thus amplifying network capabilities while saving bandwidth. The third goal is to explore appropriate decentralized RL schemes for SFCE in the context of NFV MANO. The focus here is to investigate multi-agent methods that have (as a system) a complete view of the true network state, making the overall decision-making process more reliable and grounded to reality. The final goal is to investigate SFC graph transformation techniques to improve resource efficiency in cases where available resources are limited or highly fragmented.

Overall, this thesis targets the development and application of advanced algorithms to tackle tangible, complex problems that currently plague the NFV orchestration realm, especially considering resource-constrained environments, such as edge clouds. By devising innovative SFCE strategies, exploring the potential of deep multi-agent RL, and utilizing advanced service graph transformation techniques, we hope to foster significant advancements in the practical application of NFV.

## 1.2 Contributions

The work undertaken in this thesis tackles a range of challenges in the domain of NFV and SFC optimization. Specifically:

1. A significant contribution is the identification and proposal of innovative methods for SFCE across multiple resource dimensions, such as CPU, memory, and storage. By examining multi-dimensional mapping efficiency metrics and assessing their suitability for

heuristic and exact SFCE methods, the thesis offers a deeper understanding and innovative solutions to optimize SFCE in resource-constrained NFV and beyond.

2. Moving beyond individual service management, this research presents a cross-service communication aware SFCE heuristic that optimizes SFC placement based on both inner-component resource and communication demands, as well as the requirements of another deployed SFC that will be consumed. The introduction of a new data structure, the VNF embedding tree, revolutionizes the process of managing and optimizing SFCE in the context of cross-service communication.

3. Another pivotal contribution is the proposition of a cooperative deep multi-agent reinforcement learning (DMARL) scheme for decentralized SFCE. This scheme fosters efficient communication between neighboring intelligent agents, addressing the limitations of centralized RL schemes. The behaviors learned by the team of agents are explored, providing valuable insights into how a team of RL agents operates in SFCE.

4. Lastly, this thesis contributes to the field by introducing the concept of SFC graph transformation (SFC-GT), advocating for the decomposition of VNFs into multiple instances with lower resource demands, thereby facilitating their placement onto the NFV infrastructure. By treating SFC-GT as a multi-objective optimization problem and designing a mixed-integer linear program (MILP) to solve it, notable resource efficiency gains are achieved.

Through these contributions, this thesis advances both the theoretical understanding and practical application of NFV, offering nuanced insights into the complexities of SFCE and presenting innovative solutions to a wide range of emerging NFV challenges.

## 1.3 Outline

The remainder of the thesis consists of a background chapter, followed by four technical chapters, and a conclusions chapter. In the following, we discuss the content of each chapter in more detail.

Chapter 2 serves as a primer on the very fundamental concepts and constructs that are used throughout the thesis. The discussion commences by delving into the various classes of resource allocation challenges in NFV, emphasizing their intricacies and providing concrete examples. Having established the positioning of SFCE within the overall landscape of resource allocation problems in NFV, the chapter provides formal mathematical definitions and models, which form the basis of most SFCE methodologies in the subsequent technical chapters.

Chapter 3 delves into optimizing SFCE across multiple resource dimensions. It first investigates multi-dimensional vector distance metrics and assesses their suitability for SFCE. It then discusses the design and evaluation of exact and heuristic SFCE algorithms which integrate multi-dimensional notions, and compares their performance with single-dimensional counterparts. Last, a VNF bundling scheme is examined as a means to improved balance in server resource utilization and consolidation.

In Chapter 4, we investigate cross-service communication constraints in the context of SFCE. The chapter introduces the concept of an innovative data structure, namely the VNF embedding tree, which is used to optimize the VNF embedding sequence. In the development of a cross-service communication-aware heuristic, we pay special attention to the optimization of both intra- and inter-SFC links.

Chapter 5 sets out to address the limitations of centralized RL schemes for SFCE. To this end, the chapter proposes a multi-agent framework which is better aligned to standard NFV MANO systems. Subsequently, the multi-agent framework is analyzed and evaluated against a state-of-the-art centralized RL method. The proposed distributed method fosters cross-agent communications, and reveals the impact of information exchange across various network topologies and imperfect communication channels.

Chapter 6 advocates for SFC graph transformations (SFC-GT) in the event of highly utilized and/or fragmented (resource-wise) networks. In this respect, it lays the foundations of SFC-GT by introducing the notions of embedding flexibility and transformation complexity as two (primarily contradicting) factors that need to be balanced. Then, it models SFC-GT as a mixed integer linear program by applying multiple complex linearization techniques to inherently non-linear constraints.

Finally, Chapter 7 lays out the key takeaways of the thesis, and it discusses applications and future extensions of the proposed methods.

## 1.4 Publications

The main body of this thesis is rooted in five key publications. These papers and articles specifically delve into the challenges and innovations in orchestrating SFCs within resource-constrained NFV environments. They present an exhaustive investigation starting from the fundamental aspects of multi-dimensional SFCE, progressing to cross-service communication considerations, then applying advanced deep multi-agent RL algorithms, and finally exploring the *transformation* aspect of SFC graphs for enhanced resource efficiency. The list of these core publications is as follows:

1. **Angelos Pentelas**, George Papathanail, Ioakeim Fotoglou, Panagiotis Papadimitriou. "Network service embedding with multiple resource dimensions", *IFIP/IEEE Network Operations and Management Symposium*, 2020 [9]
2. **Angelos Pentelas**, George Papathanail, Ioakeim Fotoglou, Panagiotis Papadimitriou. "Network service embedding across multiple resource dimensions", *IEEE Transactions on Network and Service Management*, vol. 18, pp. 209-223, 2020 [10]
3. **Angelos Pentelas**, Panagiotis Papadimitriou. "Network service embedding for cross-service communication", *IFIP/IEEE International Symposium on Integrated Network Management*, 2021 [11]
4. **Angelos Pentelas**, Danny De Vleeschauwer, Chia-Yu Chang, Koen De Schepper, Panagiotis Papadimitriou. "Deep multi-agent reinforcement learning with minimal cross-agent communication for SFC partitioning", *IEEE Access*, vol. 11, pp. 40384 - 40398, 2023 [12]

5. **Angelos Pentelas**, Panagiotis Papadimitriou. ”Service function chain graph transformation for enhanced resource efficiency in NFV”, *IFIP Networking Conference*, 2021 [13]

This thesis is complemented by several other studies in the field of NFV orchestration. While these publications are not a direct part of the thesis, they still relate to the overall research area. They showcase the breadth of our investigations and involvement towards a holistic and comprehensive understanding of NFV. Below is a list of these additional publications:

1. George Papathanail, Ioakeim Fotoglou, Christos Demertzis, **Angelos Pentelas**, Kyriakos Sgouromitis, Panagiotis Papadimitriou, Dimitrios Spatharakis, Ioannis Dimolitsas, Dimitrios Dechouniotis, Symeon Papavassiliou. ”COSMOS: An orchestration framework for smart computation offloading in edge clouds”, *IFIP/IEEE Network Operations and Management Symposium*, 2020 [14]

2. Ioakeim Fotoglou, George Papathanail, **Angelos Pentelas**, Panagiotis Papadimitriou, Vasileios Theodorou, Dimitrios Dechouniotis, Symeon Papavassiliou. ”Towards cross-slice communication for enhanced service delivery at the network edge”, *IEEE Conference on Network Softwarization*, 2020 [15]

3. George Papathanail, **Angelos Pentelas**, Ioakeim Fotoglou, Panagiotis Papadimitriou, Konstantinos V Katsaros, Vasileios Theodorou, Sergios Soursos, Dimitrios Spatharakis, Ioannis Dimolitsas, Marios Avgeris, Dimitrios Dechouniotis, Symeon Papavassiliou. ”MESON: Optimized cross-slice communication for edge computing”, *IEEE Communications Magazine*, 2020 [16]

4. Lucian Beraldo, **Angelos Pentelas**, Fábio Luciano Verdi, Panagiotis Papadimitriou, Cesar AC Marcondes. ”Tenant-oriented resource optimization for cloud network slicing with performance guarantees”, *IEEE Conference on Network Softwarization*, 2021 [17]

5. Margarita Bitzi, Maria-Evgenia Xezonaki, Vasileios Theodorou, Ioannis Dimolitsas, Dimitrios Dechouniotis, Symeon Papavassiliou, George Papathanail, Ioakeim Fotoglou, **Angelos Pentelas**, Panagiotis Papadimitriou. ”MESON: Optimized cross-slice communica-

tion for pervasive virtual CDN services", *IEEE International Mediterranean Conference on Communications and Networking*, 2021 [18]

6. George Papathanail, **Angelos Pentelas**, Panagiotis Papadimitriou. "Towards fine-grained resource allocation in NFV infrastructures", *IEEE Global Communications Conference*, 2021 [19]

7. Dimitrios Laskaratos, Ioannis Dimolitsas, George Papathanail, Maria-Evgenia Xezonaki, **Angelos Pentelas**, Vasileios Theodorou, Dimitrios Dechouniotis, Theodoros Bozios, Panagiotis Papadimitriou, Symeon Papavassiliou. "MESON: A platform for optimized cross-slice communication on edge computing infrastructures", *IEEE Access*, vol. 10, pp. 49322-49336, 2022 [20]

# Chapter 2

# Background

This section serves as a primer on technologies, concepts, and constructs that are central to the subject of the thesis. Specifically, we commence with a brief introduction to Network Function Virtualization (NFV), along with the standard reference architecture that has been proposed to facilitate it. Subsequently, we highlight the various management and orchestration challenges that go - unavoidably - hand-in-hand with NFV, with a particular focus on resource allocation problems. Finally, we elaborate on a core NFV resource allocation challenge, namely *service function chain embedding*, and discuss its scope, aspects, and variants, as well as the core properties of its standard form.

## 2.1 From middleboxes to virtual network functions

Firewalls, intrusion detection systems, load balancers, and network address translators are common examples of network processing appliances, typically referred to as *network functions* or *middleboxes*. Although their existence is hardly ever perceptible by the end users of the Internet, middleboxes lie at the core of every network infrastructure in order to improve security and performance, to increase availability, and to facilitate connectivity. Indeed, one of the very first surveys - conducted by *Sherry et al.* [1] - on middlebox deployments over enterprise networks reveals that the magnitude of the former is on par with the number of routers and switches of the latter, effectively ranging from a few tens (in small networks) up to tens of thousands (in very large networks). However, middleboxes are implemented as physical devices - typically resembling network switches - which in itself poses a severe challenge to infrastructure providers (InPs), especially those operating large networks. Concretely, *middleboxes are hard to scale*, since their acquisition, installation, and configuration require high capital and operational expenditures, a good deal of physical effort, as well as high personnel expertise. As a result, most middlebox deployments are provisioned for peak loads, effectively wasting most of their processing capacity during their entire lifespan.

This inherently inelastic approach to middlebox deployments is addressed via NFV [2], which emerges as a cloud computing paradigm that promotes the virtualization of middleboxes, *i.e.,* their implementation in the form of virtual machines or containers. As such, virtual network functions (VNFs) can be deployed on commodity servers of typical datacenters; by leveraging existing virtualization technologies, VNFs can now easily scale their processing capacity in response to volatile loads. Consequently, NFV opens up opportunities for on-demand network functions provisioning, thus rendering NFV-enabled networks more agile, resilient, and cost-effective.

To fill up its promise, NFV is backed by a hierarchical management and orchestration (MANO) architecture, which is illustrated in Fig. 2.1. Conceptually, at the bottom lies the Physical Resources layer, which encompasses physical servers, links, switches, base stations, etc. The Virtual Infrastructure Management (VIM) layer lies right above the Physical Resources layer, and it applies an abstraction to physical resources, essentially through virtualization. Hence, servers, links and RAN resources, for instance, can be now further divided into smaller independent segments, allowing for more fine-grained resource allocation. Such segments are realized in the form of virtual machines, containers, virtual links, etc. Last, the NFV Orchestration (NFVO) layer is positioned above the VIM layer, and it operates on an even higher level of abstraction. Concretely, the NFVO acknowledges service elements in the form of VNFs. For example, in Fig. 2.1, the NFVO view includes a service chain comprising a load balancer VNF, a Firewall VNF, and a Proxy server VNF (which is a common service chain at the application layer, *e.g.,* load balancing and monitoring flows before they reach a Web server).

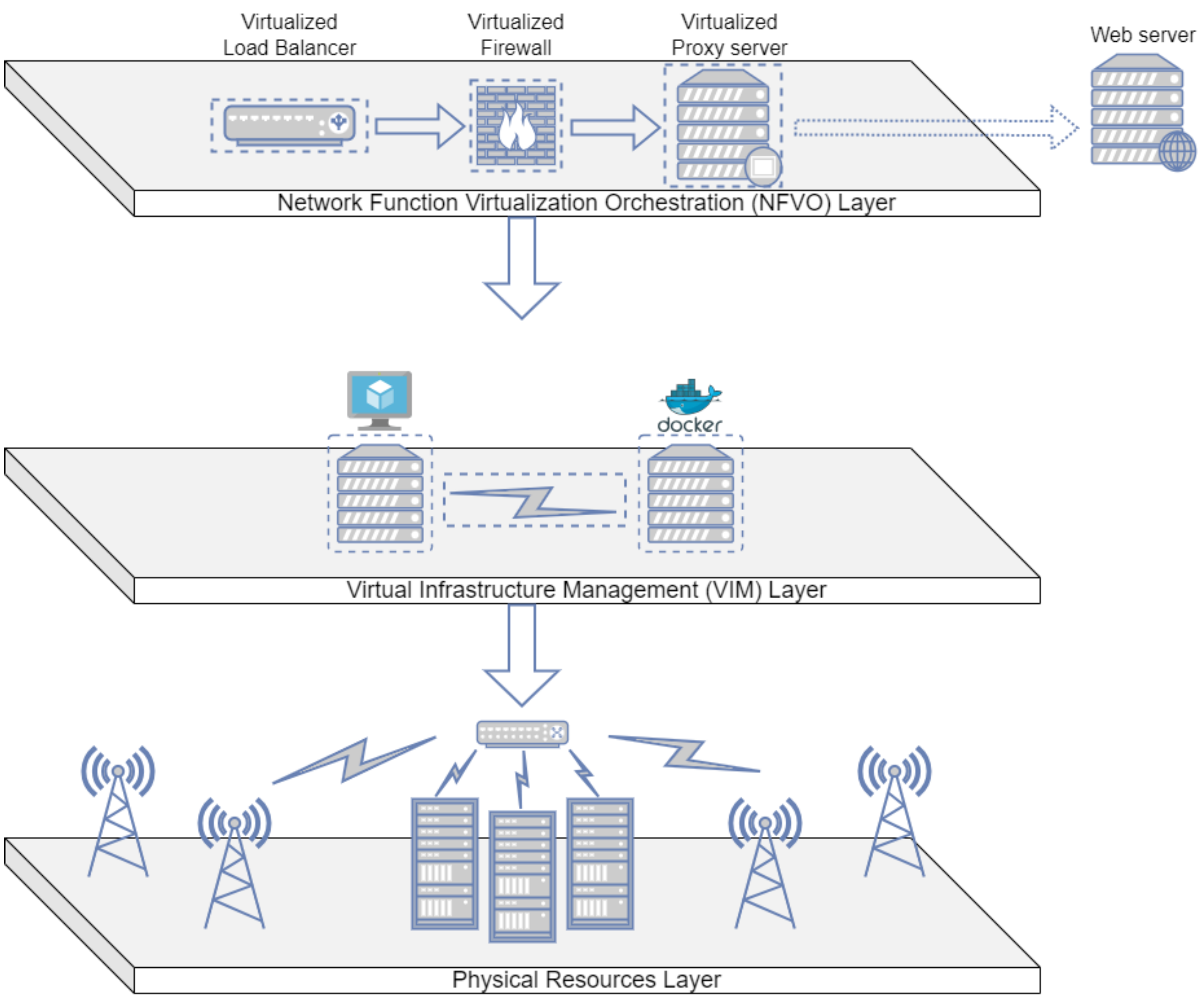

Virtualized Load Balancer
Virtualized Firewall
Virtualized Proxy server
Web server
Network Function Virtualization Orchestration (NFVO) Layer
docker
Virtual Infrastructure Management (VIM) Layer
Physical Resources Layer

Figure 2.1: The conceptual layers of an NFV MANO system.

The scope of NFV MANO is extensive, covering various MANO aspects of VNFs. NFV MANO is responsible for the lifecycle management of virtualized resources, including the onboarding, instantiation, scaling, healing, and termination of VNFs. It also encompasses the allocation and optimization of resources such as compute, storage, and network bandwidth to ensure efficient utilization and performance. Moreover, NFV MANO plays a crucial role in fault management, performance monitoring, and service chaining, enabling the detection and resolution of issues, ensuring optimal performance, and enabling the composition of multiple VNFs to create end-to-end services. Nowadays, advanced NFV MANO implementations involve policy management, automation, analytics, and intelligence, providing a holistic framework for managing and optimizing virtualized network infrastructures.

As explained in the previous discussion, the NFVO collaborates closely with VIMs, which are responsible for managing the virtual infrastructure layer that hosts the VNFs. The VIMs handle the provisioning, monitoring, and control of virtualized resources, including virtual machines, storage, and network connectivity. NFV MANO relies on the capabilities provided by VIMs to ensure the smooth operation of VNFs. This interplay between NFVO and VIMs involves communication and coordination through well-defined interfaces and protocols. NFV MANO sends resource requests and receives resource status updates from VIMs, enabling it to allocate and optimize resources effectively based on the needs and demands of the network services. The close collaboration between NFVO and VIMs is essential for achieving efficient resource management and orchestration in virtualized network environments.

## 2.2 Orchestration challenges in NFV

NFV MANO tackles several orchestration challenges to enhance the performance and efficiency of virtualized network infrastructures. Some notable challenges related to this thesis are:

**Service orchestration.** Service orchestration involves managing the deployment and interconnection of multiple VNFs to deliver complex network services. The challenge lies in defining the service topology, which includes determining the sequence of VNF instantiation and their interdependencies. Coordinating the deployment of VNFs from different vendors adds complexity due to variations in their deployment models and interfaces. Ensuring proper connectivity and configuration across the service chain is crucial to ensure end-to-end service functionality.

**Resource orchestration.** Resource orchestration focuses on efficient allocation and management of the underlying physical and virtual resources required for VNF deployment. It involves coordinating compute, storage, and networking resources to fulfill the resource demands of VNFs. Challenges include optimizing resource utilization to achieve cost-efficiency and performance, handling resource conflicts or constraints, and dynamically reallocating resources based on changing service demands. Resource orchestration also involves monitoring resource availability and capacity planning.

**Policy and performance management.** Policy and performance management in NFV

MANO involves defining and enforcing policies related to service deployment, resource allocation, and performance optimization. Policies can include quality of service (QoS) requirements, security policies, and regulatory compliance rules. The challenge lies in ensuring policy compliance across the NFV infrastructure, coordinating policy enforcement across multiple VNF instances, and adapting policies dynamically to changing network conditions. Performance management includes monitoring and analyzing key performance indicators (KPIs) such as latency, throughput, and availability, and taking appropriate actions to maintain service levels and optimize resource utilization. It further involves collecting and analyzing performance data from VNFs, VIMs, and other network elements.

Addressing these challenges requires robust orchestration frameworks, standardized interfaces, and coordination mechanisms. Standards such as the Network Service Descriptor and Virtual Network Function Descriptor specifications defined by ETSI [6] help in specifying the service and resource requirements, while standardized APIs like the NFV Orchestrator (NFVO) API [21] enable interoperability between different MANO components. Open-source projects like ONAP [8] and OpenStack [22] provide platforms and tools that assist in addressing these challenges and promoting standardization and collaboration within the NFV ecosystem.

## 2.3 Resource allocation problems in NFV

Since its original conception, NFV has expanded its scope beyond the virtualization of traditional middleboxes. Concretely, application areas pertaining to Industry 4.0, such as autonomous vehicles, remote healthcare, and smart manufacturing, require seamless, secure, and prompt network connectivity, as well as elastic networks in order to keep the overlying services reliable and affordable. The corresponding technologies which facilitate the above, *i.e.,* vehicle-to-everything (V2X) communication networks [23, 24], network slicing for ultra reliable low latency communications (URLLC) [25, 26], and edge computing [5, 16] - their common denominator being that they are fueled by NFV - have become extremely vibrant fields of research, effectively attracting researchers and practitioners across a wide spectrum of both academia and industry. Besides Industry 4.0, NFV also empowers the virtualization of modern mobile cores, such as 5G (and beyond) deployments. Here, both the control plane functions (*e.g.,* access and management mobility function - AMMF, session management function - SMF, policy control function - PCF, etc.) and the data plane functions (*e.g.,* user plane functions - UPFs) are implemented as VNFs. This enables mobile network operators to scale their 5G deployments in response to volatile load, and to bring data processing closer to the RAN for more efficient network utilization.

However, deploying NFV-enabled networks is far from trivial. Although the magnitude of physical effort and the risk of experts-driven deployments (which can turn out quite error prone, especially considering the load stochasticity exhibited by modern networks) are limited with NFV, it is now software that comes to the spotlight. Concretely, this software needs to handle a wide range of operations, including the configuration, the deployment, the instantiation, the monitoring, the scaling, and the repositioning of VNFs, to name a few. Since the scope of this thesis touches upon the algorithmic challenges of NFV, we will refrain from diving into pure technical problems. In fact, we will only focus on a subset of NFV algorithmic challenges, namely NFV Resource Allocation (NFV-RA) problems [4]. As the domain of interest is relatively new and, thereby, constantly evolving, we will attempt to provide our own classification of the most common NFV-RA problems. Our taxonomy has its grounds on systematic encounters in literature, our exposure in the related industry, as well as our own interpretations. As such, Fig. 2.2 should not be taken as definitive, rather as a reference point that facilitates subsequent discussions.

### 2.3.1 Static vs dynamic problems

We commence our discussion by elaborating on what we deem as a crucial property of each NFV-RA problem we have encountered thus far, namely the way it treats *the effects of time.*

A *static* NFV-RA problem is effectively one whose scope disregards the potential evolution of network parameters that might occur over time. That is, static NFV-RA problems typically work with problem instances which are formed by a snapshot of the problem's environment.

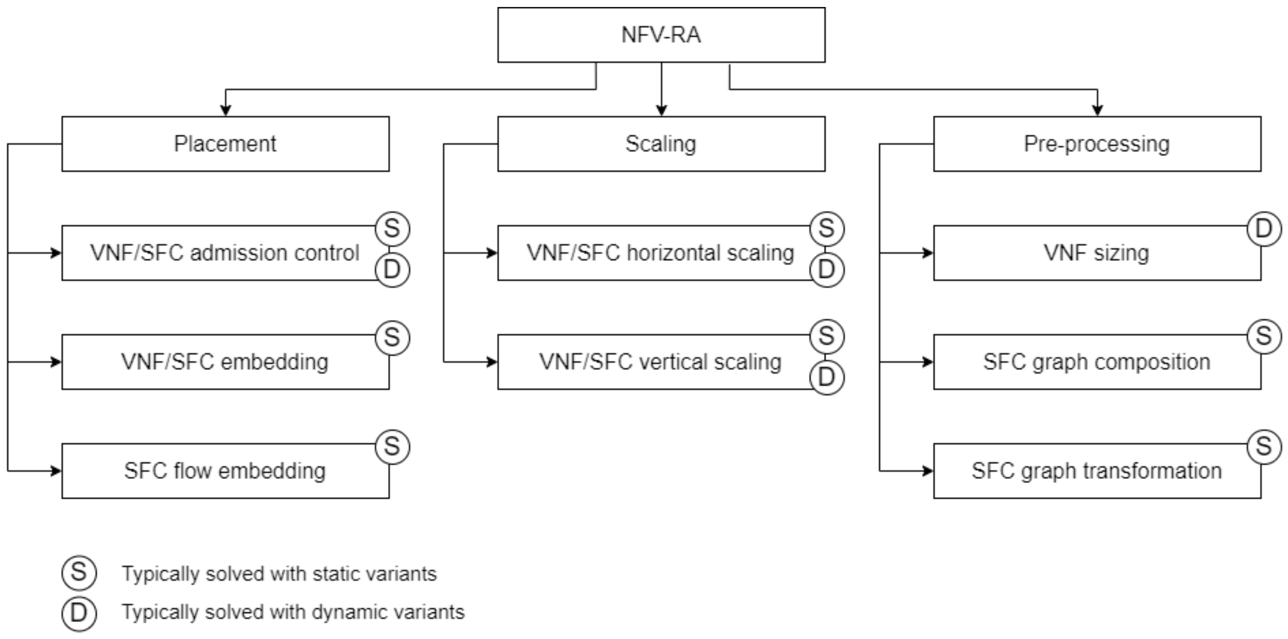

NFV-RA
Placement
Scaling
Pre-processing
VNF/SFC admission control S D
VNF/SFC embedding S
SFC flow embedding S
VNF/SFC horizontal scaling S D
VNF/SFC vertical scaling S D
VNF sizing D
SFC graph composition S
SFC graph transformation S
S Typically solved with static variants
D Typically solved with dynamic variants

Figure 2.2: A taxonomy of common resource allocation challenges in NFV.

That is because the decision horizon of static NFV-RA problems is too narrow for parameter updates to be critical. For instance, a static VNF admission control problem has a decision horizon in the order of double-digit milliseconds, thus we can safely assume that the parameters of the problem do not vary significantly within such short time-windows. For example, if a VNF cannot be admitted to a datacenter because of insufficient resources *now*, it will not be able to be admitted for the same reason *after* 20ms as well. Naturally, this rationale might not always hold since the boundaries between correct and wrong assumptions (like the one above) can sometimes be blurry. In fact, static NFV-RA problems can be seen as a simplification of their rather *dynamic* counterparts. Yet, it is more convenient to frame a problem as static, since this allows for easier mathematical modelling and harnessing of off-the-shelf optimization solutions, such as mixed-integer linear programs (MILPs) and heuristics, which are - in most cases - simple to design, use, and explain.

In contrast to static NFV-RA problems, *dynamic* ones position the evolution of network parameters at their epicenter. Here, the scope of the decision horizon extends to wider time-windows (sometimes, even infinite) comprising multiple decision steps, thus it is essential to keep track of the transitions of the problem's environment. A dynamic NFV-RA problem can be seen as a sequence of repetitions of its static counterpart; in effect, one could utilize the solver of the latter at every decision step of the former (since each individual step is the entire horizon of the static version). In this case, though, the optimality of the *end-to-end solution*[1] is not guaranteed most of the time. In principle, modelling dynamic NFV-RA problems is complex - to be more precise, modelling the evolution of parameters accurately is the most complex part - and solving them to optimality is usually either intractable or computationally inefficient. Nevertheless, this does not mean that there can be no simplifications. For example, the VNF (vertical) scaling problem - which acknowledges a dynamic environment by definition - can be modelled in a reactive fashion, and then solved by a threshold-based optimization policy, *e.g.,* if the observed load of the VNF exceeds 80%, increase its core frequency by a predetermined value; if the observed load drops below 50%, decrease its core frequency accordingly.

As will be further clarified from the discussions in Sections 2.3.2 - 2.3.4, the distinction into static and dynamic is not an inherent characteristic of an NFV-RA problem. Instead, it characterizes *the approach to modelling and solving* the problem. That is, there are many cases where the very same NFV-RA problem can be approached with static, dynamic, and even hybrid variants. As a rule, the more the dynamic aspects of an approach to a problem - that is, the more the emphasis on learning and modelling the (distributions of the) parameters of the problem, and on the downstream effects of decisions on its environment - the more accurately the problem is solved. This makes an approach particularly appealing to researchers. In contrast, a more static framing of a problem makes it more myopic, but less complex also. This makes it particularly appealing to engineers and production systems. Indicatively, this is corroborated by the highly rich literature on sophisticated data-driven VNF scaling schemes; yet, state-of-the-art NFV MANO systems, such as Open Source MANO [7] and Open Baton [27], employ simple threshold-based scaling policies. With that in mind, there are no specific guidelines on how to decide the level of dynamicity when tackling an NFV-RA problem. As such, we deem that the solutions proposed in this thesis lie somewhere in-between the two extremes, *i.e.,* we aim at both modelling accuracy and practicality[2].

* * *

[1]We use the term *end-to-end solution* to refer to the union of single-step solutions.

[2]Practicality refers to, *e.g.,* solvability, explainability, deterministic performance, etc.

We will now briefly discuss several NFV-RA problems which stand out in the respective literature. This will allow the reader to better grasp the landscape of resource allocation challenges within NFV and, in extension, the positioning of this thesis. Importantly, we will attempt to identify the level of dynamicity that is typically employed to solve (the standard form of) each problem, based on the above discussions. Recall that the term SFC stands for service function chain. For now, it suffices to know that an SFC is an ordered sequence of VNFs, which are grouped together to enforce an elaborate, end-to-end flow processing policy (cf. top of Fig. 2.1). A more formal definition of SFCs follows in Section 2.4.

### 2.3.2 Placement problems

NFV-RA placement problems focus on finding the best possible positions of NFV constructs (such as VNFs, SFCs, and SFC flows[3]) onto physical network infrastructures. The prevailing way of modelling placement problems usually involves the definition of a *request model* and a *network model.* Commonly, NFV elements are associated with performance objectives, which are translated into geographical constraints and resource requirements; such parameters are captured by the request model. In contrast, network models formalize resource availability, *i.e.,* absolute and/or relative position of a resource, as well as its capacity. For example, a VNF placement request can be associated with CPU requirements and geographical constraints, while the underlying physical network can model a collection of servers scattered within a geographical area and offering CPU resources. Eminent placement problems include:

**VNF/SFC admission control.** Admission control mechanisms can be seen through the lens of NFV-RA placement problems, in the sense that they decide whether a VNF/SFC can be placed within an infrastructure[4] or not. The static version of these problems is rather trivial, since the solver only needs to ensure that resource constraints can be satisfied (this is practically a constraint satisfaction problem, instead of an optimization one). However, the VNF/SFC admission control problem can be rendered quite complex in its dynamic form, since the solver now needs to compute the downstream effects of admitting each VNF/SFC over time

[3]An SFC flow consists in a collection of network packets which are to traverse the VNFs of the SFC.
[4]Infrastructure is vaguely defined here, *e.g.,* it can be an entire network, a datacenter, or a single server.

(*e.g.,* whether the admission of a VNF *now* might prevent the admission of other - potentially more valuable[5]- VNFs in the *future*). Admission control mechanisms in the context of NFV placement problems can be found in [28, 29, 30].

**VNF/SFC embedding (SFCE).** SFCE lies at the heart of NFV placement challenges. Effectively, solvers that address these problems need to optimize mappings of virtual elements (pertaining to the request model) to physical elements (pertaining to the network model), subject to resource and location constraints. The objectives here may vary; the most common are latency minimization, VNF collocation (alternatively, server consolidation), and load balancing. Irrespective of the objective, these optimization problems - even in their simplest, static form - are NP-hard. As such, exact algorithms (*e.g.,* MILPs) are only used in small-scale instances, while problems of medium and large size are mostly solved via heuristics or approximation methods. Dynamic variants of SFCE are scarce in the literature, as the problem becomes (practically) intractable. We will elaborate further on the complexity aspects of SFCE in Section 2.4.3. Nonetheless, it is worth to note that SFCE exhibits many similarities with the well-known virtual network embedding (VNE) problem; hence, it is not an overstatement to argue that the literature on SFCE has been built upon its VNE counterpart. Important works that tackle the problem at hand are, *e.g.,* [31, 32, 33].

**SFC flow embedding.** The SFC flow embedding problem can be seen as a rather user-centric variant of the SFCE problem, since the prevailing objective here is to minimize flow latency (which is perceptible by the end-user). Concretely, given a set of VNFs which are *already embedded* within a network, the problem of SFC flow embedding boils down to the computation of a subset of these VNFs such that i) the selected VNFs implement the correct end-to-end flow processing policy as specified by the SFC, ii) resources of both VNFs and the network suffice to serve the flow, and iii) the flow is steered correctly and efficiently. These problems typically require shortest-path algorithms (*e.g.,* Dijkstra's method) and elaborate heuristics to compute the best VNF subset. Because of its inherent complexity, SFC flow embedding is hardly ever formalized in a strict dynamic form. Instead, the literature approximates its dynamicity with static versions which, *e.g.,* account for SFC flows batching. For further information, we refer

[5]It is common to assume that a VNF admission returns some form of reward to the network provider.

the interested reader to [34].

* * *

Taking a more holistic view, the above problems are all logically connected: at first, an admission control mechanism decides whether a VNF/SFC can be placed within an infrastructure; then, an embedding algorithm is employed to compute the optimized position of the VNF/SFC. Finally, an SFC flow embedding method computes the VNFs and the network paths each flow will need to traverse. Indeed, the literature exhibits a few studies which target more than one of the above problems at the same time (*e.g.,* [28, 35, 36]). This supports our intuition of classifying them under the same problem category, namely NFV-RA placement problems.

### 2.3.3 Scaling problems

As mentioned in Section 2.1, NFV fosters the virtualization of network functions, which greatly enhances their scaling capacity. This in itself is extremely crucial, since NFV workloads exhibit high stochasticity and are, admittedly, difficult to predict. Prior to NFV, network operators had to plan ahead of time (quarterly, biannually, annually, etc.) the size of their middlebox deployments (*e.g.,* the number of additional firewalls they will need), according to projections of future loads[6]. If we link that to previous discussions, we will agree that this is essentially a dynamic NFV-RA problem, since the decision horizon considers the evolution of the *load* parameter. However, this inherently dynamic challenge was traditionally approximated with a robust, yet inefficient, policy; that is, operators used to size middleboxes for peak loads. Naturally, this resulted in over-provisioned infrastructures, which wasted energy and resources during off-peak periods, *i.e.,* a substantial amount of time. This changed drastically with NFV.

Loosely defined, VNF scaling is the practice of adding or removing compute or network capacity to a VNF in response to volatile demands. Examples of compute resource types are CPU, memory, and GPU, while common network resource types are bandwidth and DPDK-enabled

[6]We use the term load to capture multiple metrics, such as CPU load, number of sessions, data traffic, etc.

ports [37]. The very nature of a VNF allows it to leverage virtualization techniques to scale *in* or *out* and *up* or *down*, a process that can last from a few milliseconds (*i.e.,* scaling up/down a VNF implemented as container - CNF) up to a few minutes (*e.g.,* scaling out a VNF implemented as virtual machine - VM). Given the fact that the VNF scaling duration is now practically negligible (compared to the duration required for purchasing a middlebox, having a network technician approach the installation location, and connect the device to the network), the original middlebox sizing problem can be dealt with more efficiently. Concretely, the problem can be now approximated with two efficient variants: a static variant, namely *reactive* scaling (*e.g.,* [38]), and a dynamic variant, namely *proactive* scaling (*e.g.,* [39]). As per the former, VNFs scale in response to observed loads; with the latter, the solver predicts the evolution of loads to perform the ideal scaling ahead of time. Both reactive and proactive scaling can be realized via the following practices:

**Horizontal scaling.** According to horizontal scaling, a VNF scales in or out by adding or removing VNF instances, respectively. This is the prevailing way of VNF scaling nowadays, and it is natively supported by most VIMs, such as Openstack and Kubernetes. A critical observation here is the following: while scaling out a VNF, new VNF instances need to be spawned. These instances are not yet placed anywhere in the physical network, thereby an embedding mechanism is still required to compute their optimized physical positions. This further stresses the importance of the SFCE problem.

**Vertical scaling.** A VNF scales vertically when resources are added to (scale up) or removed from (scale down) a running instance. The most viable way of applying vertical scaling is by varying the frequency of the CPU cores assigned to the VNF instance, or by varying its last-level cache (LLC) share. However, these features require advanced hardware technology, such as Intel® RDT [40, 41], which is supported only by modern CPUs of general-purpose servers. Consequently, vertical scaling is applied on a less frequent basis compared to horizontal scaling.

* * *

Irrespective of the scaling practice (*i.e.,* horizontal or vertical) and the scaling variant (*i.e.,* reactive or proactive), the core challenge of the VNF scaling problem is to compute the exact resources required for given or predicted load values. The risk at hand is that, if the assigned resources are more than required, then the operator is exposed to unnecessary costs, while if resources are less than actually needed, then the performance of the VNF might drop. These issues can be handled efficiently by offline scaling frameworks, which presume a VNF profiling process. Profiling records the performance behavior of the VNF under varying loads and resources, thus enabling the corresponding scaling mechanism to make (near-)optimal decisions. In contrast, online scaling schemes learn efficient loads-resources combinations on-the-fly, with the downside of exploratory actions during the early stages of learning.

### 2.3.4 Pre-processing problems

We have intentionally left VNF/SFC pre-processing problems last, since these can be seen as rather complementary to the ones already mentioned. That is, most VNF/SFC pre-processing problems emerge to augment the - as already hinted - quite complex placement and scaling problems described above. The key-element here is to pre-process the loosely defined parts of the request model in a way that facilitates the solver of the main problem to either obtain more effective solutions or to obtain solutions more efficiently.

**VNF sizing.** In Section 2.3.3, we established how naive middlebox sizing strategies can evolve into sophisticated VNF scaling policies through NFV. Thus, one might argue that VNF sizing (*i.e.,* deciding the number of additional VNF instances which will be needed in a network infrastructure) is no longer an issue, as it can be dealt with in a timely fashion using horizontal scaling mechanisms. While this is true for the most part, in practice a VNF sizing step is still required. That is because several VNFs are developed by third-party organizations (*e.g.,* telecommunications software vendors), and network operators need to purchase licenses of these VNFs in order to use them on their production environments. Therefore, the VNF sizing problem now comes down to computing the right number of VNF licenses to purchase. Nonetheless, flexible pricing schemes, which are widely adopted in the cloud computing domain, can be applied here as well, essentially limiting the adverse impact of potentially inaccurate

VNF sizing.

In a sense, VNF sizing augments horizontal scaling solvers by providing an upper bound to the maximum number of VNF instances. A notable work that employs VNF sizing is [42].

**SFC graph composition.** The SFC graph (alternatively, the VNF forwarding graph - VNF-FG) composition problem centers on re-ordering the VNFs of an SFC in order to compose an improved graph alternative. The objectives here are usually either (processing) latency or (average or maximum) bandwidth minimization. Naturally, SFC graph composition can only be applied on SFCs whose structure is relatively loose; that is, VNF re-orderings do not modify the original flow processing policy of the SFC which was specified by the user. For instance, in a *source* → *firewall* → *load balancer* → *destination* SFC, the two VNFs can exchange positions without affecting the flow processing logic. However, the final sequence can have a crucial impact on the SFC performance.

Undoubtedly, SFC graph composition is tightly linked to the SFCE problem, since VNF re-orderings alter the request model and its corresponding requirements (*e.g.,* bandwidth requirements of virtual links between VNF pairs). A more formal definition, as well as an interesting solution to this problem can be found in [43].

**SFC graph transformation.** SFC graph transformation (SFC-GT) formulates the problem of decomposing the VNFs of an SFC into multiple instances, in order to augment an SFCE solver compute feasible solutions even in cases of high resource fragmentation. The latter refers to the problem of highly distributed resource availability (*e.g.,* a datacenter can offer ten CPU cores, yet these are available across ten servers), which is quite common in all kinds of datacenters (*e.g.,* both cloud and telco) and can lead to request admission rejections. SFC-GT considers two critical factors when deciding how to transform an SFC: the *embedding flexibility* of the SFC graph, and its *transformation complexity*. SFC-GT strives to find the sweet-spot between highly flexible, yet low complex transformations, taking into account the given resource fragmentation levels of the underlying network.

As already hinted, SFC-GT can be seen a quite useful pre-processing tool for the SFCE problem.

SFC-GT is one of the main contributions of this thesis, and it will be detailed in Chapter 6.

* * *

In some sense, the thesis touches upon all three aforementioned problem categories, either explicitly or implicitly. Concretely, we put great emphasis on placement problems and, in particular, on SFCE problems. The main reasons for that are the following:

1. SFCE is constantly introduced to new complexity dimensions. That is, both the request model, as well as the network model, are highly affected by developments on Internet technologies, such as emerging Industry 4.0 applications, the virtualization of the mobile core and RAN, and innovative network slicing concepts. Consequently, existing approaches can become obsolete or ineffective extremely quickly, and new investigations are needed to address each variant with tailored solutions.

2. SFCE lies at the intersection of NFV placement and scaling problems. Indeed, our discussion in Section 2.3.3 illustrates how horizontal scaling is augmented by SFCE methods. This further stresses the importance of efficient SFCE solvers.

3. Emerging *zero-touch network automation* [44] requirements which have been set out by network vendors and operators pave the way for the investigation and the development of innovative, data-driven SFCE algorithms, primarily based on machine learning techniques. As will be evident in Chapter 5, we pay particular attention to interpreting such methods, as a way to enhance the practicality of the solution.

4. Pre-processing techniques for SFCE problems have not been thoroughly examined, which is apparent in the (lack of) respective literature. This opens up the opportunity to design appropriate SFC pre-processing frameworks, taking into account practical issues of modern network infrastructures, such as resource fragmentation.

## 2.4 The service function chain embedding problem

This section is centered on the SFCE problem. Specifically, it aims to i) formally define important concepts which are extensively used throughout the thesis, ii) demonstrate the very essence and practical challenges of SFCE through examples and illustrations, and iii) substantiate the reasons which render SFCE a problem that requires special algorithmic treatment via its computational complexity analysis. Thereby, this section serves as an SFCE reference point within the thesis.

### 2.4.1 Formal definitions and network models

**Definition 2.1** (SFC)**.** An SFC $A$ consists in an set of VNFs $V_A$, and a strict partial order $\prec_A$ defined over $V_A$. That is, $A = (V_A, \prec_A)$, and $V_A$ is a partially ordered set. Given two VNFs $i, j \in V_A$, if $i$ precedes $j$ we write $i \prec_A j$. □

In effect, the precedence relationship ($\prec_.$) orders most VNF pairs belonging to the VNF set of an SFC. This ordering is essential for the functionality of the SFC. Concretely:

**Definition 2.2** (SFC policy)**.** Given an SFC $A = (V_A, \prec_A)$, the strict partial order $\prec_A$ dictates the exact VNF sequences that can be traversed by each network flow of $A$. These VNF sequences comprise the policy of $A$. □

The strict partial order of each SFC can be defined explicitly (via explicit ordering of all VNFs) or implicitly (via relative ordering). Example 2.1 illustrates the derivation of an SFC policy through implicit VNF orderings.

**Example 2.1.** *A network operator deploys an SFC A where the set of VNFs is $V_A$= {firewall (FW), intrusion detection system (IDS), load balancer (LB)}. The operator wants flows to traverse i) the IDS prior to the FW and the LB, and ii) the FW prior to the LB. That is, IDS $\prec_A$ FW $\prec_A$ LB, and the (policy of) SFC $A = (V_A, \prec_A)$ can be pictured as:*

$$A = \mathit{source} \rightarrow IDS \rightarrow FW \rightarrow LB \rightarrow \mathit{destination}$$

□

Notice that, in Example 2.1, all VNF pairs in $V_A$ are ordered on the grounds of $\prec_A$. SFCs with this property form a linear chain. Specifically:

**Definition 2.3** (Linear SFC)**.** An SFC $A = (V_A, \prec_A)$ is linear iff $i \prec_A j$ or $j \prec_A i$, $\forall i, j \in V_A,\ i \neq j$. □

The policy of linear SFCs consists in a unique VNF sequence. However, there are cases where an SFC policy comprises more than one VNF sequences, as shown in Example 2.2.

**Example 2.2.** *A network operator deploys an SFC B where the set of VNFs is $V_B$= {firewall (FW), intrusion detection system (IDS), load balancer (LB), network address translation (NAT), }. The operator wants flows to traverse the FW prior to any other VNF. Flows that pass the FW control successfully can move on to the LB and then the NAT. Flows that are rejected by the FW are forwarded to the IDS for further inspection. That is, FW $\prec_B$ LB $\prec_B$ NAT, and FW $\prec_B$ IDS. Thus, the policy of SFC $B = (V_B, \prec_B)$ consists of two valid VNF sequences:*

$$B_1 = source \rightarrow FW \rightarrow LB \rightarrow NAT \rightarrow destination_1$$

*and*

$$B_2 = source \rightarrow FW \rightarrow IDS \rightarrow destination_2$$

*Here, $destination_2$ can be a server of the network operator that logs intrusion attempts.* □

Apparently, in Example 2.2, not all VNF pairs in $V_B$ are $\prec_B$-comparable. In particular, we cannot compare neither the $LB$ nor the $NAT$ to the $IDS$ on the grounds of $\prec_B$. SFCs such as $B$ are termed non-linear. In general:

**Definition 2.4** (Non-linear SFC)**.** An SFC $A = (V_A, \prec_A)$ is non-linear iff $\exists\ i, j \in V_A,\ i \neq j$ and $i, j$ $\prec_A$-incomparable. □

While Definitions 2.1 and 2.2 suffice for both linear and non-linear SFCs, the thesis focuses on linear cases. This decision has its grounds on the following realization: non-linear SFCs can be eventually decomposed into multiple linear ones, where each individual VNF sequence represents the policy of a linear SFC.

In the remainder of the thesis, we might as well define SFCs in expressive forms, such as:

$$A = \{\text{VNF}_1 \prec_A \text{VNF}_2 \prec_A ... \prec_A \text{VNF}_k\} \text{ or } A = \text{VNF}_1 \rightarrow \text{VNF}_2 \rightarrow ... \rightarrow \text{VNF}_k$$

where the VNF set $V_A$ and the strict partial order $\prec_A$ are both clear.

* * *

As mentioned in Section 2.3.2, most NFV-RA placement problems are modelled with the help of a request model and a physical network model. SFCE is not an exception. In the following, we formally define these two models, and we elaborate on how SFCE is built upon them. Notice that the terms *physical network* and *substrate* are used interchangeably throughout the text.

**Definition 2.5** (Request model)**.** An SFC is modelled with a directed graph $G = (V, E)$. The set of vertices $V$ comprises the VNFs of the SFC, while the set of edges $E$ consists of the virtual links (*i.e.,* conceptual connections) among adjacent VNFs. Each VNF is associated with resource demands across a finite set of resources $R$, and the demand of VNF $i \in V$ on resource $r \in R$ is denoted by $d_r(i)$. Each virtual link $(i, j) \in E$ has bandwidth demands, which are expressed as $d(i, j)$. □

Admittedly, considering the Definitions 2.1 and 2.2, as well as the expressive forms of SFCs, it is somewhat intuitive that SFCs are mathematically modelled as directed graphs. An instance of a request model is given in Example 2.3.

**Example 2.3.** *Consider an SFC consisting of VNFs $V = \{f_1, f_2\}$ and $E = \{(f_1, f_2)\}$. Then, $G = (V, E)$ can be pictured as:*

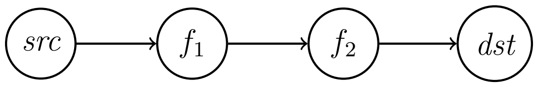


Figure 2.3: Illustration of a linear SFC with two VNFs.

*Further, let $R = \{CPU\}$. If the VNF $f_1$ requires 10 CPU cores, we would write $d_{CPU}(f_1) = 10$, and if the $(f_1, f_2)$ virtual link requires $700Mpbs$ of bandwidth, we would write $d(f_1, f_2) = 700$.* □

**Definition 2.6** (Substrate model)**.** A substrate is modelled with an undirected graph $G_S = (V_S, E_S)$. The set of vertices $V$ comprises compute and routing nodes, and the set of edges $E_S$ consists of the physical links among adjacent nodes. The set of resources offered by the compute nodes of the substrate are denoted with $R_S$. Each node $u \in V_S$ offers a limited amount of each resource type $r \in R_S$, which is denoted with $c_r(u)$. Each physical link $(u, v) \in E_S$ has a limited bandwidth capacity, which is expressed as $c(u, v)$. Further, $P(u, v)$ denotes all paths that connect two nodes $u, v \in V_S$, and $P(E_S)$ comprises all paths in $E_S$. □

**Example 2.4.** *Consider a substrate comprising $V_S = \{n_1, n_2, n_3, n_4, n_5\}$ and $E_S = \{(n_1, n_2), (n_1, n_3), (n_2, n_3), (n_3, n_4), (n_3, n_5)\}$. Then, $G_S = (V_S, E_S)$ can be drawn as:*

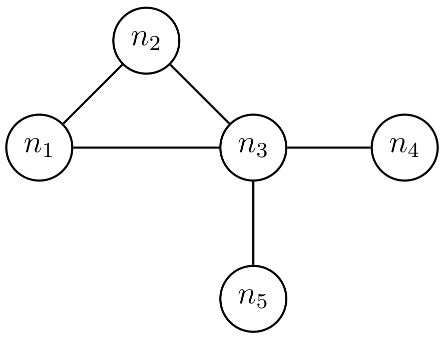


Figure 2.4: Illustration of a substrate network comprising five nodes.

*Further, let $R_S = \{CPU, GPU\}$. If node $n_1$ offers 64 CPU cores, we would write $c_{CPU}(n_1) = 64$. If $n_1$ offers no GPU, we would write $c_{GPU}(n_1) = 0$. If the $(n_1, n_2)$ physical link offers $1000Mpbs$ of bandwidth, we would write $c(n_1, n_2) = 1000$. Here, $P(n_1, n_3) = \{[(n_1, n_3)], [(n_1, n_2), (n_2, n_3)]\}$.* □

Apparently, $R \subseteq R_S$, as it only makes sense that the VNFs of the SFC request resources that are indeed offered by the substrate. As such, we might as well write simply $R$ to denote the common set of resource types across both SFC and substrate models.

**Definition 2.7** (Feasible node embedding)**.** Given an SFC request $G = (V, E)$ and a substrate $G_S = (V_S, E_S)$, an SFC node embedding is feasible iff there exists a function $m_V : V \mapsto V_S$ such that $\{m_V(i) = v \mid v \text{ can accommodate } i,\ \forall i \in V,\ v \in V_S\}$. □

**Definition 2.8** (Feasible edge embedding)**.** Given an SFC request $G = (V, E)$ and a substrate $G_S = (V_S, E_S)$, an SFC edge embedding is feasible iff there exists a function $m_E : E \mapsto P(E_S)$ such that $\{m_E(i, j) = P(u, v) \mid (u, v)$ can accommodate $(i, j),\ \forall(i, j) \in E,\ (u, v) \in P(u, v),\ P(u, v) \in P(E_S)\}$. □

Arguably, in Definitions 2.7 and 2.8, the term *can accommodate* is somewhat vague. In the standard SFCE form, it primarily refers to the satisfaction of resource constraints. However, in more complex SFCE settings, further practical aspects can be considered, such as location proximity constraints, affinity and anti-affinity rules, volume bindings, etc.

**Definition 2.9** (Feasible SFCE)**.** Given an SFC request $G = (V, E)$ and a substrate $G_S = (V_S, E_S)$, an SFCE is feasible iff there exists a feasible node embedding dictated by $m_V$ and a feasible edge embedding dictated by $m_E$. A feasible SFCE is denoted by the tuple $m_G = (m_V, m_E)$, and the set of all feasible SFCEs is expressed with $M_G$. □

In the context of SFCE, each feasible SFCE is quantified in accordance with a specified optimization objective. Specifically:

**Definition 2.10** (SFCE fitness)**.** Given an SFC request $G = (V, E)$, a substrate $G_S = (V_S, E_S)$, and a feasible SFCE $m_G = (m_V, m_E)$, the fitness of $m_G$ is determined by a function $h : M_G \mapsto \mathbb{R}$. Without loss of generality, we assume that the goal is to minimize $h$. □

**Example 2.5.** *Consider the SFC request and the substrate from Examples 2.3 and 2.4, respectively. We assume that the src (source) of the SFC is associated with the physical node $n_1$, while the dst (destination) of the SFC is associated with the physical node $n_3$. In practice, these initial assignments are not under the control of an SFCE mechanism, and are therefore considered parameters instead of variables of the problem. For the sake of simplicity, we assume that physical elements can accommodate all of their virtual counterparts (e.g., resources always suffice). Along these lines, there can be a fitness function $h_1$ which quantifies the quality of an embedding on the grounds of how many physical nodes are used by the VNFs of the SFC. As such, the embedding depicted on the left ($m_G^{left}$) is better than the embedding on the right ($m_G^{right}$), since the former uses one physical node (i.e., $n_2$), while the latter uses* *two (i.e., $n_1$ and $n_3$). Hence, $h_1(m_G^{left}) = 1 < 2 = h_1(m_G^{right})$. Similarly, a different fitness function $h_2$ can quantify the embedding quality based on the minimum number of hops between the src and the dst, while traversing the VNFs in the correct order. As per $h_2$, we have that $h_2(m_G^{right}) = 1 < 2 = h_2(m_G^{left})$, and $m_G^{right}$ is better than $m_G^{left}$. That is, flows need to traverse both $(n_1, n_2)$ and $(n_2, n_3)$ according to $m_G^{left}$, while they only need to utilize the edge $(n_1, n_3)$ according to $m_G^{right}$.*

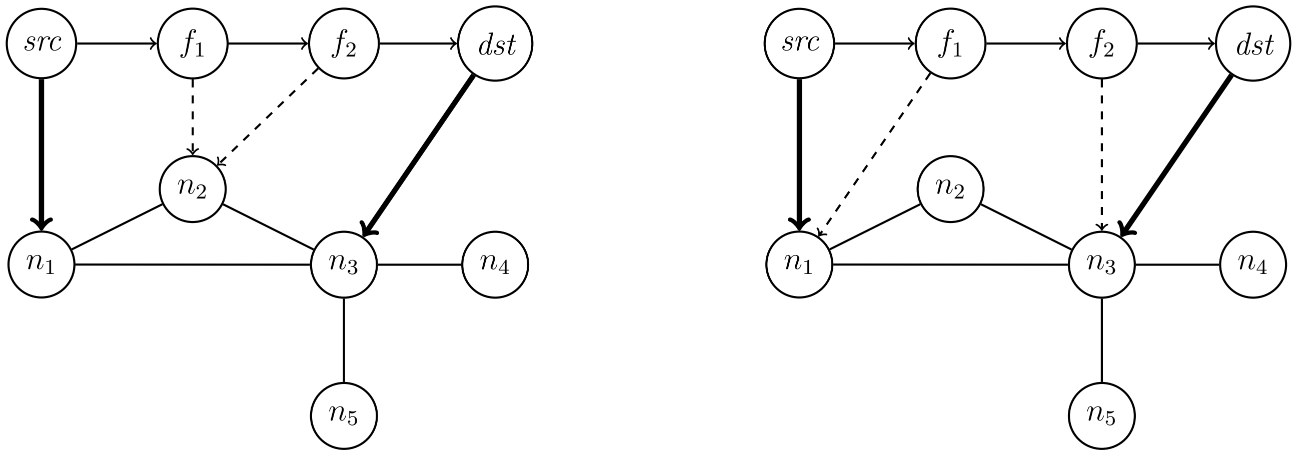


Figure 2.5: In terms of VNF collocation, the SFCE on the left is preferred, whereas in terms of hop count, the SFCE on the right is superior.

□

**Definition 2.11** (SFCE)**.** Given an SFC request $G = (V, E)$, a substrate $G_S = (V_S, E_S)$, and an arbitrary optimization objective quantified by a fitness function $h$, SFCE consists in the computation of a feasible SFCE that minimizes $h$. Concretely, the goal is to:

$$\min_{m_G \in M_G} h(m_G)$$

□

In simpler terms, SFCE boils down to computing the *best* (according to a specified optimization objective) feasible mapping between the virtual elements of the request model, and the physical elements of the substrate model. The last definition is about the decision counterpart of SFCE, which will be mainly used in the theoretical analysis of SFCE in Section 2.4.3.

**Definition 2.12** (D-SFCE)**.** Given an SFC request $G = (V, E)$ and a substrate $G_S = (V_S, E_S)$, D-SFCE strives to answer if a feasible SFCE $m_G = (m_V, m_E)$ exists. □

### 2.4.2 Practical aspects of SFCE and optimization objectives

The practical aspects and the optimization objectives of SFCE can vary based on the type of the substrate, which can take two main forms: datacenter network or multi point-of-presence (multi-PoP) network. The following paragraphs discuss how these two variants affect the scope and the properties of SFCE.

**Datacenter substrate.** A datacenter network $G_S$ is a collection of interconnected servers and switches; these are the compute and routing nodes ($V_S$) of the substrate, respectively. Servers are typically organized in racks. Different servers of the same rack exchange data through intra-rack links, using a top-of-the rack (ToR) switch. For instance, *server-1* and *server-2* in Fig. 2.6 communicate via links $(1, A)$ and $(2, A)$. Servers that lie in different racks utilize inter-rack links for their communication, which is facilitated by aggregation (AG) switches. As an example, the connection of *server-1* and *server-3* utilizes links $(1, A)$, $(A, B)$, $(B, C)$, and $(C, 3)$. Intra- and inter-rack links comprise the set of substrate edges $E_S$. Notice that AG switches bring certain racks together, hence, in some sense, certain pairs of servers belonging to different racks are closer compared to others. We showed, for example, that *server-1* and *server-3* are four hops away; however, the communication of *server-1* and *server-4* requires a more lengthy path consisting of six hops, *i.e.,* $(1, A)$, $(A, B)$, $(B, D)$, $(D, E)$, $(E, F)$, and $(F, 4)$. The network depicted in Fig. 2.6 is essentially a three-layer fat-tree datacenter, which is a quite common datacenter topology.

With regards to the problem at hand, it is quite often that network operators, vendors, or enterprises want to deploy SFCs within a single datacenter network. In these cases, it is assumed that the collocation of the VNFs (in a spatial sense) is more favourable compared to the case where the VNFs would have spanned multiple datacenters (this is the case of the multi-PoP substrate). In the context of a single-datacenter, SFCE examines two critical dimensions: i) *improved SFC performance* and ii) *resource utilization efficiency of the substrate.* Improved SFC performance is typically ensured via dense VNF collocation. In more detail, as we witnessed in the previous paragraph, there is always a path between two different servers of a datacenter. However, the length of this path can vary. Since VNFs are effectively deployed

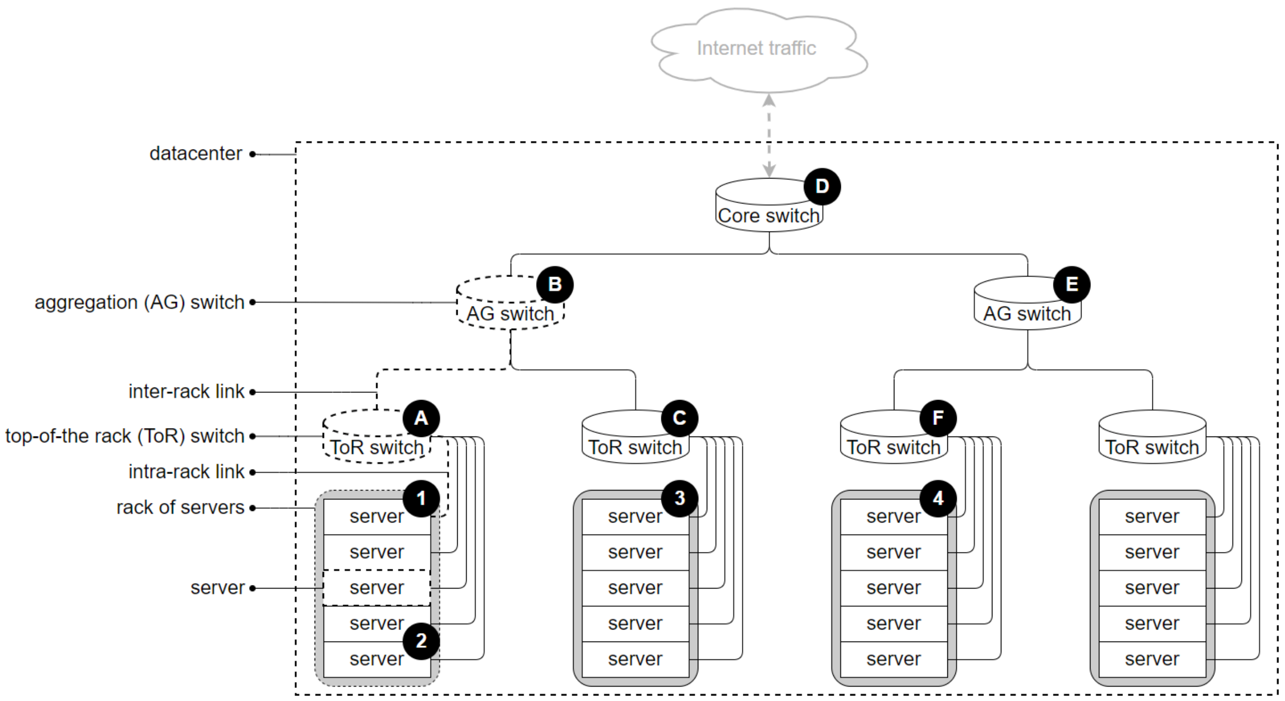


Figure 2.6: Illustration of a three-layer fat-tree datacenter network.

on servers, it makes sense that VNFs which continually communicate are positioned in servers which are in close proximity with one another, since them being distant can result in extensive communication delay (hence, performance degradation). The following example discusses three alternative SFCEs and evaluates them on the grounds of VNF collocation.

**Example 2.6.** *Let $G = (V, E)$ be the SFC of Example 2.3. Notice that we can ignore the src and dst nodes while modelling the SFC, since these are usually associated with locations beyond the datacenter-level scope. Further, let us consider a simplified subset of the substrate illustrated in Fig. 2.6, namely $G_S = (V_S, E_S)$, where $V_S = \{1, 2, 3, 4\}$ and $E_S = \{(u, v), \forall u, v \in V_S, u \neq v\}$. In the graphs below, the weights on the edges represent the length of the path between the corresponding servers. For example, a path from server-1 to server-3 uses 4 physical links. Further, each graph expresses a different SFCE. The SFCE on the left ($m_G^{left}$) maps both VNFs on server-1, the SFCE in the middle ($m_G^{middle}$) maps $f_1$ on server-1 and $f_2$ on server-2, while the SFCE on the right ($m_G^{right}$) assigns $f_1$ to server-1 and $f_2$ to server-3. In terms of VNF collocation, $m_G^{left}$ is better than $m_G^{middle}$, and $m_G^{middle}$ is better than $m_G^{right}$. That is, assuming that $h$ is a fitness function that quantifies the VNF collocation efficiency, we have that $h(m_G^{left}) = 0$, since the $(f_1, f_2)$ virtual link does not use any physical links*[7]*, $h(m_G^{middle}) = 2$ and $h(m_G^{right}) = 4$.*

[7]Actually, $(f_1, f_2)$ is confined within *server-1*, and virtual switching is employed to establish the communi-

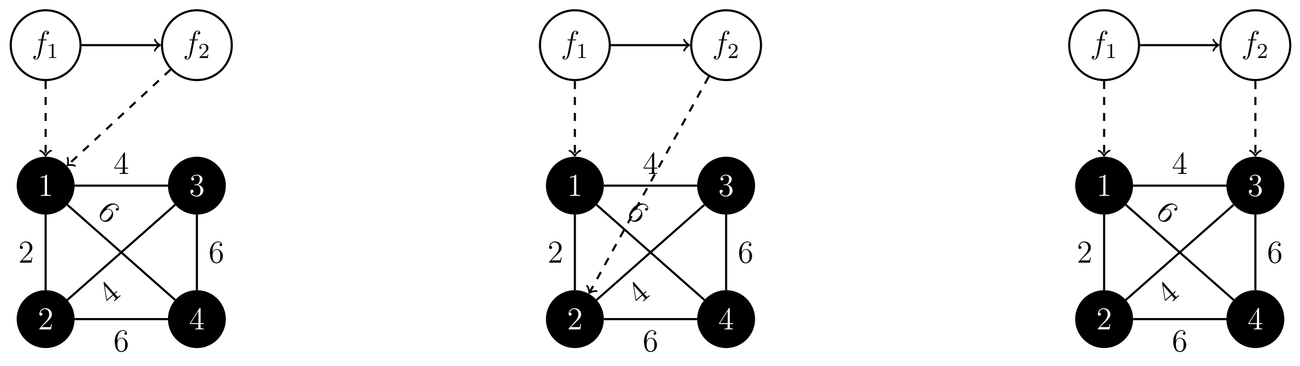


Figure 2.7: In terms of VNF collocation, the SFCE on the left is preferred, since it maps the $(f_1, f_2)$ link within a single server, thereby establishing fast communication between the VNFs $f_1$ and $f_2$.

□

We established how VNF collocation is an SFCE criterion that regards the performance of a single SFC. However, a datacenter can host multiple SFCs, as well as additional applications which do not necessarily pertain to NFV. As such, the datacenter operator needs to take a more holistic view of their substrate, and to devise resource allocation plans that satisfy the requirements of a diverse set of services. This brings us to the second dimension examined by the datacenter-level SFCE, namely the *resource utilization efficiency of the substrate.* Here, common goals are *server consolidation* and *server load-balancing.* The former refers to the utilization of a minimized amount of servers, such that the unutilized ones can be shut down, thus saving power. Naturally, server consolidation is inline with VNF collocation, and the two objectives can be combined easily. However, there are cases where VNF-to-server assignments might compromise VNF collocation in favour of improved long-term resource utilization (*e.g.,* minimization of resource fragmentation). In contrast, server load-balancing aims at keeping server utilization at equal levels; this way, the risk of server malfunctions due to resource saturation is minimized. Apparently, load-balancing stands against VNF collocation, and the two objectives are, in some sense, conflicting. Importantly, both notions can be naturally extended to physical link resources as well. That is, server consolidation leads naturally to link consolidation, thus saving switch TCAM resources which further reduces power consumption, while server load-balancing leads to link load-balancing, where congestion situations can be

cation between $f_1$ and $f_2$.

easily avoided.

**Example 2.7.** *Consider the embeddings illustrated in Example 2.6. In terms of server consolidation, $m_G^{left}$ remains the best option. However, on the grounds of load-balancing, $m_G^{middle}$ and $m_G^{right}$ seem better alternatives. In fact, an SFCE that would target both VNF collocation and server load-balancing would prefer $m_G^{middle}$, as a compromise between the two objectives.* □

***

The discussion above unveils some practical aspects of SFCE over a single datacenter, it exemplifies its most common objectives, and it corroborates its intricacies. Now, we will delve into a different SFCE scope where the substrate consists in a network of datacenters, instead of a single datacenter.

**Multi-PoP substrate.** A multi-PoP network is a collection of interconnected PoPs. Each PoP represents a geographical location where the provider has infrastructure, such as servers, routers, and switches. Throughout the thesis, we will assume that each PoP is a datacenter network, as defined and described earlier. Fig. 2.8 illustrates a hypothetical multi-PoP substrate $G_S$ comprising 11 PoPs over Europe; these are the nodes $V_S$ of the substrate. Further, adjacent PoPs are connected via inter-PoP links, which are the edges $E_S$ of the substrate.

SFCE over multi-PoP networks deals with deciding the optimal assignment of VNFs to the underlying PoPs. For *improved SFC performance*, operators aim at VNF collocation (similar to SFCE over a single datacenter), since service latency can be highly perceptible in a multi-PoP scale. Hence, placement strategies typically consider the latency minimization objective. Factors that can contribute towards latency-efficient SFCEs in this context are VNF collocation, coupled with the distribution of VNFs across short paths from the *src* to the *dst* of the SFC. Conversely, for the *resource utilization efficiency of the substrate*, operators commonly account for throughput maximization or load-balancing. To achieve throughput maximization, operators need to carefully assess the bandwidth requirements of multiple SFCs which are (to be) hosted within their network, and place them in a way such that inter-PoP links are utilized in an optimal fashion. This is on par with latency minimization under the assumption that links

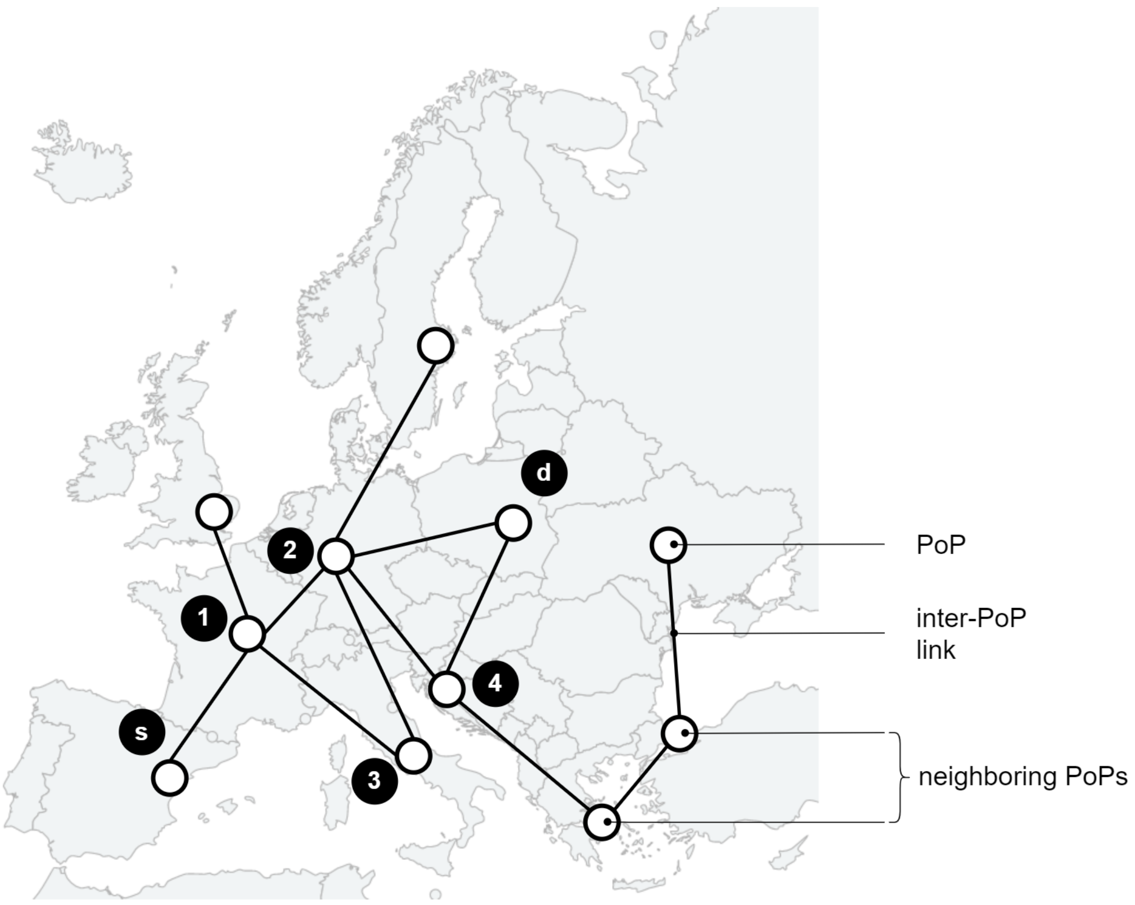


Figure 2.8: Illustration of a multi-PoP topology spanning Europe.

are never congested, which would have an adverse effect on latency. Load-balancing extends the notions of the single datacenter case in the multi-PoP setting, thus it remains conflicting with the objective of latency minimization.

**Example 2.8.** *Let $G = (V, E)$ be the SFC of Example 2.3. This time we cannot ignore the src and dst nodes of the SFC, since their positioning plays a crucial role in determining the SFCE alternatives. Specifically, according to Fig. 2.8, there are* 4 *possible (acyclic) paths from node s to node d. These are:*

*$path_1$:* $(s, 1) \rightarrow (1, 2) \rightarrow (2, d)$

*$path_2$:* $(s, 1) \rightarrow (1, 3) \rightarrow (3, 2) \rightarrow (2, d)$

*$path_3$:* $(s, 1) \rightarrow (1, 2) \rightarrow (2, 4) \rightarrow (4, d)$

*$path_4$:* $(s, 1) \rightarrow (1, 3) \rightarrow (3, 2) \rightarrow (2, 4) \rightarrow (4, d)$

*Assuming that all edges incur the same latency, some of these paths are apparently shorter* *than others. For instance,* $path_1$ *is shorter than* $path_2$ *and* $path_3$*, which are both shorter than* $path_4$*. However, even though* $path_4$ *seems an unreasonable choice for mapping the SFC of Example 2.3, it still can be a viable option under special circumstances. For example, if the substrate is resource-wise saturated and nodes* $s$*,* $1$*,* $2$*, and* $d$ *cannot accommodate either* $f_1$ *or* $f_2$*, then these might need to be placed in nodes* $3$ *and* $4$*, respectively.*

□

### 2.4.3 Complexity of SFCE

The SFCE problem is known to be NP-hard [45], and finding an optimal embedding solution becomes challenging as the scale and complexity of the network increase. In particular, the computational complexity of the SFCE stems from several practical factors:

**Combinatorial nature.** The SFCE problem is inherently combinatorial in nature. It involves mapping both VNF instances and their corresponding virtual links to the available physical or virtual resources while satisfying various constraints. As the number of VNFs, virtual links, and available resources increases, the number of possible embedding combinations grows exponentially, leading to an exponential search space.

**Resource constraints.** The embedding problem needs to consider various resource constraints, including CPU, memory, bandwidth, and latency requirements of the VNFs, as well as the available capacity of the underlying infrastructure. Balancing resource utilization across the infrastructure and ensuring that the resource requirements of the VNFs are met adds complexity to the problem.

**Inter-dependency considerations.** SFCs often have inter-dependencies between the VNFs, where the output of one VNF serves as the input to another. Embedding these inter-dependent VNFs while maintaining the correct order of their execution becomes a complex task. Ensuring proper connectivity and data flow between the VNFs within the chain introduces additional computational challenges.

**Dynamic and real-time constraints.** The SFCE problem becomes more complex when dynamic and real-time constraints are considered. Network conditions, traffic patterns, and service demands may change over time, requiring the embedding to be adaptable and flexible. Efficiently adapting the embedding to meet changing demands while minimizing disruptions to the existing service instances poses computational challenges.

Due to the computational complexity of the SFCE problem, finding an optimal solution in a reasonable amount of time is often infeasible. As a result, heuristic and approximation algorithms are commonly employed to address this challenge. These algorithms aim to find near-optimal or suboptimal solutions by using techniques like constraint relaxation, greedy algorithms, genetic algorithms, and machine learning-based approaches.

For the sake of completeness, we show that the decision counterpart of a relatively simple SFCE variant is NP-complete.

**Theorem 2.1.** *D-SFCE with multiple resource types and zero bandwidth demands of virtual links is NP-complete.*

*Proof.* We perform a polynomial-time reduction to the NP-complete Multidimensional Bin Packing decision problem.

*Simplified SFCE Problem.* Given a set of servers $S = \{S_1, ..., S_m\}$, each with certain capacities in $r$ dimensions (*e.g.,* CPU, memory, bandwidth), and a set of VNFs $V = \{V_1, ..., V_n\}$, each requiring certain resources in $r$ dimensions, the problem is to determine whether there exists a mapping of each VNF to a server such that no server's capacity is exceeded in any dimension.

*Multidimensional Bin Packing Problem.* Given a set of $r$-dimensional items $I = \{I_1, ..., I_n\}$ and a set of bins $B = \{B_1, ..., B_m\}$, each with a certain capacity in $r$ dimensions, the problem is to decide whether all items can be packed into the bins such that the total size of the items in each bin along each dimension does not exceed the bin's capacity in that dimension.

*Reduction.* We construct an instance of the Multidimensional Bin Packing problem from an instance of the SFCE problem as follows. Each server $S_i$ in the SFCE problem corresponds to a bin $B_i$ in the Multidimensional Bin Packing problem, with the capacity of $B_i$ in each dimension set equal to the capacity of $S_i$ in that dimension. Each VNF $V_j$ in the SFCE problem corresponds to an item $I_j$ in the Multidimensional Bin Packing problem, with the size of $I_j$ in each dimension set equal to the resource requirements of $V_j$ in that dimension. Under this construction, a solution to the SFCE problem (a mapping of VNFs to servers such that no server's capacity is exceeded for any resource type) corresponds directly to a solution to the Multidimensional Bin Packing problem (a placement of items into bins such that no bin's capacity is exceeded in any dimension).

This reduction can be performed in polynomial time, which means the simplified SFCE problem is at least as hard as the Multidimensional Bin Packing problem. Thus, we can conclude that the simplified SFCE problem is NP-complete. $\square$

# Chapter 3

# Service Function Chain Embedding across Multiple Resource Dimensions

As NFV expands to cater to the needs of virtualized mobile networks and emerging 5G (and beyond) services, the resource demands across dimensions like CPU, memory, and storage become more diverse. This increase in requirements presents a challenge for SFCE, as most existing methods primarily consider a single resource dimension, typically the CPU, making them inefficient. This chapter explores methods to optimize SFCE across multiple resource dimensions. We examine various multi-dimensional mapping efficiency metrics and evaluate their suitability for both heuristic and exact SFCE solvers. By leveraging the most suitable and efficient metrics, we propose two heuristics and a mixed integer linear program (MILP) for optimized multi-dimensional SFCE. Additionally, we introduce a VNF bundling scheme that generates balanced VNF bundles to enhance VNF placement. Our evaluation demonstrates significant gains in resource efficiency using the proposed heuristics compared to single-dimensional approaches, along with a slight sub-optimality compared to the proposed MILP. Furthermore, we explore the impact of the bundling scheme on embedding efficiency when combined with our most efficient heuristic. Further, this chapter uncovers valuable insights and potential implications of employing multi-dimensional metrics in SFCE methods.

## 3.1 Motivation

Many proposed methods for SFCE face a notable limitation, as they are designed to account for a single resource dimension. However, this restriction proves to be highly problematic, particularly when considering nodes (*i.e.,* VNFs). In practical scenarios, VNF resource demands must be expressed across multiple dimensions, including CPU, memory, and storage. As the scope of NFV continues to expand, encompassing virtualized cellular network elements [46, 3] and specialized service elements [47], a wide diversity of requirements is expected to emerge across these resource dimensions. Focusing solely on CPU demand, which is a common practice in the majority of SFCE methods, for VNFs with intensive memory or storage needs can lead to significant resource wastage. Consequently, this resource inefficiency translates into potential revenue loss for the InP. The implications of relying on single-dimensional SFCE methods become even more severe in resource-constrained NFV environments, such as edge clouds, where resource efficiency is of paramount importance.

To overcome these challenges, it is crucial to develop SFCE methods that consider multiple resource dimensions simultaneously. By accounting for the unique resource demands of VNFs across CPU, memory, and storage, InPs can achieve a more efficient and optimized resource allocation. This comprehensive approach not only minimizes resource waste but also ensures that the varying requirements of different VNFs are adequately addressed.

In light of these, this chapter aims to bridge the gap in SFCE methods by proposing innovative techniques that account for multi-dimensional resource demands. By integrating advanced optimization algorithms and heuristics, we seek to enhance the resource efficiency and performance of SFCE in diverse NFV environments. Through extensive evaluations and comparisons, we aim to demonstrate the superiority of multi-dimensional SFCE methods over their single-dimensional counterparts, particularly in terms of resource utilization, revenue generation, and overall quality of service placements.

## 3.2 Contributions

Our study aims to quantify the advantages of multi-dimensional SFCE methods compared to approaches that consider only CPU demands, and to examine the extent to which balanced resource utilization across multiple dimensions can ensure high resource efficiency. Our extensive evaluations reveal significant gains in resource efficiency when employing multi-dimensional SFCE methods. Additionally, we discover that integrating a multi-dimensional mapping metric into a heuristic is a complex process that requires careful coupling to achieve optimal efficiency. We further devise and evaluate a VNF bundling mechanism that seeks to generate VNF bundles out of individual VNFs with unbalanced resource demands. More precisely, the main contributions of this chapter are the following:

- We investigate the suitability of multi-dimensional mapping efficiency metrics for SFCE across multiple resource dimensions.
- We propose both heuristic and exact methods for the embedding of SFCs with demands across multiple resource dimensions.
- We assess the efficiency of multi-dimensional metrics for SFCE, and further quantify the resource efficiency gains compared to an SFCE method that accounts for a single resource dimension.
- We perform a comparison between our most prominent heuristic and an MILP, in terms of SFCE optimality and solver run-time.
- We propose a VNF bundling scheme and assess its efficiency by coupling it with our most efficient heuristic.
- We shed light on challenging aspects and potential implications from the utilization of multi-dimensional metrics within SFCE heuristics and exact methods.

## 3.3 Problem Description

In this section, we elaborate on how methods from vector bin packing (VBP) can be incorporated into SFCE, and further explain the integration of multi-dimensional allocation principles into a heuristic algorithm and an MILP that we later present and evaluate.

Research in the field of single-dimensional VM assignment has produced several algorithms that achieve high levels of optimization [48]. However, when dealing with multi-dimensional VMs and servers, the problem becomes considerably more complex. That is, the notion of a VM *best fitting* a server cannot be straightforwardly extended to many dimensions. In practice, the multi-dimensional VM allocation problem is approached as a variant of the VBP problem and algorithms implemented for the former draw upon research on the latter.

The VBP problem involves allocating $n$-dimensional items to $n$-dimensional bins while minimizing the number of bins used. This problem has been proven to be an NP-hard optimization problem [49]. Consequently, various heuristic algorithms have been developed as more suitable approaches for solving it, such as [48, 50, 51, 52]. In our case, if we associate the items with VNFs and the bins with servers, VBP algorithms can be directly applied to handle the virtual node placement phase mentioned in Section 2.4.1. However, since these algorithms cannot consider the bandwidth and VNF collocation requirements of an SFC, their success in addressing our specific problem is not guaranteed. We propose solutions for this later in this chapter.

The shortcomings of a simplistic allocation policy, which considers three resource type requirements for VNFs, are clearly demonstrated in Fig. 3.1. Essentially, if we place the topmost VNF in *Server 1*, a significant portion of its first resource type remains unused. This is because the other two resource types of the server are already fully occupied, preventing the accommodation of additional VNFs. On the contrary, placing the VNF in *Server 2* appears to be more appropriate due to the opposite reason, namely, achieving more efficient utilization of resources. Essentially, the corresponding resources marked with the bold line are implicitly bounded; however, the fact that they are not assigned to any particular VNF prevents InPs from monetizing them. Techniques addressing this problem can be deemed efficient only when

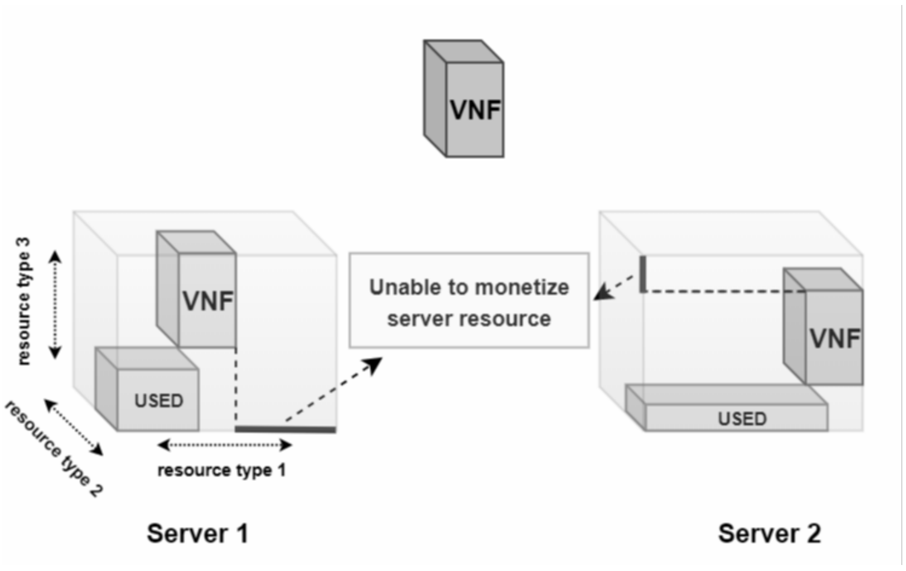


Figure 3.1: VNF-to-server mapping with *3-dimensional* nodes.

they minimize such resource wastage.

Yet, recall that SFCE focuses on SFCs which typically involve more than one VNFs. As a result, conventional VBP algorithms, which allocate items to bins based on holistic criteria, cannot be directly employed in the specific problem we are studying. Our objective is to investigate possible approaches to utilize VBP insights within the realm of SFCE. We expect that this exploration will lead to improved resource utilization in servers compared to other relevant methods that do not incorporate multi-dimensional resource mapping.

## 3.4 Methodology

This section presents a comprehensive approach for optimizing the efficiency of SFCE algorithms across multiple resource dimensions. In particular, it is structured into five subsections, namely, i) mapping efficiency metrics, where we explore methods from VBP that can be suitable for integration with SFCE solvers, ii) a mixed integer linear program (MILP) for exact SFCE, iii) a scalable multi-dimensional heuristic for near-optimal SFCE, iv) a single-dimensional heuristic that serves as a baseline, and v) a greedy VNF bundling scheme that exercises multi-dimensional notions on optimized batches of VNFs, instead of single VNFs.

### 3.4.1 Mapping efficiency metrics

Let $G = (V, E)$ be an SFC, $G_S = (V_S, E_S)$ be a datacenter substrate, and $R = \{1, \ldots, n\}$ be their common finite set of resource types. We use the vector $\mathbf{d}(i) = (d_1(i), \ldots, d_n(i))$ to indicate the demands of a VNF $i \in V$ over $R$. Similarly, we denote with $\mathbf{c}(u) = (c_1(u), \ldots, c_n(u))$ the vector of available capacities of a server $u \in V_S$. According to this notation, we now present the way each metric is computed, as well as its broader scope.

**L2 norm-based greedy.** Based on the *L2 norm-based greedy* metric, the server $u$ that hosts VNF $i$ in the most efficient way will be the one that minimizes the quantity:

$$s_u^i = \sqrt{\sum_{r=1}^{n} \Big(c_r(u) - d_r(i)\Big)^2} \tag{3.1}$$

Eq. (3.1) computes the euclidean distance between the $n$-dimensional points expressed by the coordinates of $\mathbf{c}(u)$ and $\mathbf{d}(i)$. Thus, the *L2 norm-based greedy* selects the server $u \in V_S$ that minimizes this distance.

**Dot product - Cosine similarity.** Given the vectors $\mathbf{c}(u)$ and $\mathbf{d}(i)$, their dot-product can be computed with the analytical expression, *i.e.,* Eq. (3.2):

$$\mathbf{c}(u) \cdot \mathbf{d}(i) = \sum_{r=1}^{n} c_r(u) \cdot d_r(i) \tag{3.2}$$

As per the *dot-product* metric, the most suitable server $u \in V_S$ for hosting VNF $i$ is the one that maximizes Eq. (3.2). The dot-product of the two vectors can be also calculated as:

$$\mathbf{c}(u) \cdot \mathbf{d}(i) = ||\mathbf{c}(u)|| \cdot ||\mathbf{d}(i)|| \cdot \cos\theta \tag{3.3}$$

where $||\mathbf{c}(u)||$ and $||\mathbf{d}(i)||$ indicate the norms of the respective vectors, while $\cos\theta$ expresses the cosine of their angle (which is illustrated in Fig. 3.2). Consequently, and given that $\mathbf{d}(i)$ is

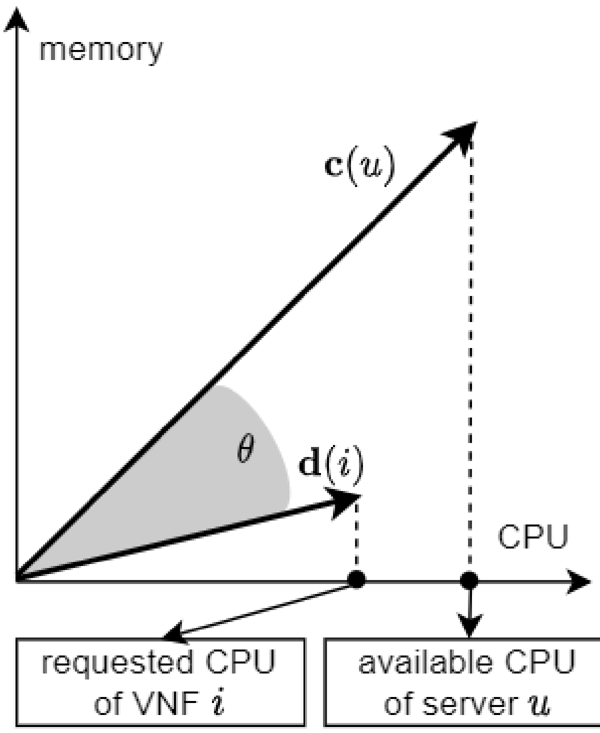


Figure 3.2: The angle $\theta$ between the vector of a server's residual capacities, $\mathbf{c}(u)$, and the vector of a VNF's resource requirements, $\mathbf{d}(i)$, for a 2-dimensional scenario.

fixed for a specific VNF $i \in V$, the dot-product in Eq. (3.3) increases if the value of $||\mathbf{c}(u)||$, $\cos\theta$ or both also increase. Nevertheless, the effect of $||\mathbf{c}(u)||$ on the final selection of a server may outweigh that of $\cos\theta$. Since we deem the latter term as more appropriate for finding aligned (according to available and required resources) server-VNF pairs, we utilize the *cosine similarity* metric, shown in Eq. (3.4):

$$s_u^i = \cos\theta = \frac{\mathbf{c}(u) \cdot \mathbf{d}(i)}{||\mathbf{c}(u)|| \cdot ||\mathbf{d}(i)||} \tag{3.4}$$

The coordinates of all vectors participating in Eq. (3.4) are positive and thus their angle lies within $[0, \frac{\pi}{2})$. Therefore, the domain of $s_u^i$ will be (0,1] and, apparently, the closer this value to one, the better.

**Resource skewness.** The notion of resource skewness is introduced in [53] as a means to quantify the fairness of the utilization of a server's various resource types. In accordance with the applicability of the other two metrics, we can compute the skewness of each server using:

$$s_u^i = \sqrt{\sum_{r=1}^{n} \Big(\frac{c_r(u) - d_r(i)}{\bar{c}(u)} - 1\Big)^2} \tag{3.5}$$

where $\bar{c}(u)$ expresses the average consumption of all resources of server $u$. In this way, Eq. (3.5) can be used to assign a cost to each VNF-to-server mapping alternative, for a specific VNF $i$ and the available servers $u$. The smaller the skewness of a server the more balanced its resource utilization is. Therefore, we would opt for the server $u \in V_S$ that minimizes Eq. (3.5).

Nevertheless, we compute skewness as shown in Eq. (3.6). Specifically, we omit the terms $d_r(i)$ from the nominators of the previous equation, and compute the average resource skewness of all servers according to their current consumption levels in order to obtain a single value for an individual datacenter. Hence, we adopt this metric as a holistic resource utilization efficiency measure.

$$\text{datacenter skewness} = \frac{\sum_{u \in V_S} \sqrt{\sum_{r=1}^{n} \big(\frac{c_r(u)}{\bar{c}(u)} - 1\big)^2}}{|V_S|} \tag{3.6}$$

**Manhattan distance.** In the MILP that we later present, we aim at incorporating a metric that acts similarly to those already presented. More specifically, this metric will act as a penalty assigned to the servers selected during the SFCE, in the sense that the more balanced their resource consumption is the better (and vice versa). It becomes apparent, though, that the metrics presented thus far are computed with non-linear terms. Hence, by incorporating them into an MILP, the element of linearity would be lost. Furthermore, the execution time of the resulting solver would be extended. To circumvent this, we incorporate the *Manhattan distance* into our MILP. According to Eq. (3.7), this distance is computed as the sum of the absolute differences of the respective coordinates of the two vectors.

$$s_u^i = \sum_{r=1}^{n} |c_r(u) - d_r(i)| \tag{3.7}$$

Despite the fact that the *Manhattan* metric seems incapable of expressing aspects, such as the angle of two vectors, its simplicity and linearity make it computationally efficient. Thus, it stands out as the most appropriate metric for our MILP (cf. Section 3.4.2). Instead, the *L2 norm-based greedy* and the *cosine similarity* metrics are incorporated into the heuristic algorithm, which is presented in Section 3.4.3.

Note that each resource dimension $r \in R$ can be assigned with a weight, by multiplying an $\alpha_r$ value with the $r$-th dimension of both $\mathbf{d}(i)$ and $\mathbf{c}(u)$. Such weights can adjust the impact of a resource type on the embedding computation. For instance, someone can assign a higher weight to the CPU dimension compared to memory, which, in turn, may be given a higher weight than storage. In our evaluations (cf. Section 3.5), such adjustments are implied by different resource scarcities, mandated by VNF resource demands and server resource capacities (*i.e.,* such weights are applied implicitly).

### 3.4.2 Mixed integer linear program

This section presents an exact SFCE algorithm based on MILP principles. While we do not anticipate that the MILP will scale favorably in our NP-hard optimization problem, it is crucial to develop and compare one against the other proposed methods in order to determine their empirical gap from theoretically optimal solutions.

Let $x_{i,u} \in \{0,1\}$ be a binary decision variable that indicates the assignment of VNF $i$ on server $u$. Similarly, let $f_{(i,j),(u,v)} \in \mathbb{R}_{\geq 0}$ be a continuous decision variable expressing the fractional mapping of the virtual link $(i,j)$ on the physical link $(u,v)$ (*i.e.,* flows can be split across multiple paths). Last, we introduce $z_u$ binary variables to express the allocation of server $u$ from the current embedding, and $M_u \in \mathbb{R}_{\geq 0}$ variables to compute the Manhattan distance between the vector of available capacities of server $u$ and (aggregated) requested capacities from the current embedding. The linear constraints of our MILP are the following:

$$\sum_{u \in V_S} x_{i,u} = 1, \quad \forall i \in V \tag{3.8}$$

$$\sum_{i \in V} x_{i,u} d_r(i) \leq c_r(u) z_u, \quad \forall r \in R, \forall u \in V_S \tag{3.9}$$

$$M_u = \sum_{r \in R} \Big( z_u c_r(u) - \sum_{i \in V} d_r(i) x_{i,u} \Big) \quad \forall u \in V_S \tag{3.10}$$

$$\sum_{\substack{v \in V_S \\ v \neq u}} \big( f_{(i,j),(u,v)} - f_{(i,j),(v,u)} \big) = x_{i,u} - x_{j,u},$$

$$\forall (i,j) \in E, \forall u \in V_S \tag{3.11}$$

$$\sum_{(i,j) \in E} f_{(i,j),(u,v)} d(i,j) \leq c(u,v), \quad \forall (u,v) \in E_S \tag{3.12}$$

$$\sum_{\substack{v \in V_S \\ v \neq u}} f_{(i,j)(u,v)} \leq 1 - x_{j,u}, \forall (i,j) \in E, \forall u \in V_S \tag{3.13}$$

$$x, z \in \{0,1\} \tag{3.14}$$

$$f, M \in \mathbb{R}_{\geq 0} \tag{3.15}$$

Practically, constraints (3.8) impose the mapping of each VNF at exactly one server, while constraints (3.9) ensure that i) servers capacities are not exceeded and ii) $x$ and $z$ variables are associated properly (*i.e.,* $z_u$ becomes 1 for $u$ is utilized, 0 otherwise - via the objective function). Constraints (3.10) assign Manhattan distance scores to each server $u$ as a means to quantify balanced embeddings. Constraints (3.11) associate $x$ and $f$ variables on the grounds of flow conservation. For example, if $x_{i,u} = x_{j,u} = 0$, then $u$ might be an intermediate node for $(i,j)$; thus, the outbound traffic of $u$ with respect to the flow $(i,j)$, which is $\sum_{v \in V_S - \{u\}} f_{(i,j),(u,v)}$, must be equal to its inbound traffic, $\sum_{v \in V_S - \{u\}} f_{(i,j),(v,u)}$. If $u$ is not visited by $(i,j)$, then both sums will be zero. A similar binding of $x$ and $f$ is performed when $u$ is source or sink node for $(i,j)$. Finally, constraints (3.12) ensure that link bandwidth capacities are not violated, constraints (3.13) avoid loops in link mappings, and constraints (3.14) and (3.15) enforce variable domains. Our objective is to:

$$\text{Minimize} \quad \sum_{u \in V_S} (z_u + \alpha M_u) + \frac{1}{\sum\limits_{(i,j) \in E} c(i,j)} \sum_{(u,v) \in E_S} \sum_{(i,j) \in E} f_{(i,j),(u,v)} \tag{3.16}$$

subject to constraints (3.8) - (3.15). In the objective function (3.16), the value of $\alpha$ acts as a weight parameter that signifies the importance of efficient VNF-to-server allocations (for a more thorough discussion on the adjustment of this parameter, see Section 3.7). For a given SFC, the objective function gives preference to solutions that utilize fewer servers efficiently (first constituent of (3.16)), while, at the same time, minimize the volume of network traffic in the datacenter (second constituent of (3.16)).

### 3.4.3 Multi-dimensional heuristic

Based on [31], we implement a heuristic that initially ranks the racks of the datacenter in descending order, according to the average available capacity of their top-of-the-rack (ToR) to core switch links (***line 5*** in Algorithm 1). The attempt of embedding an SFC within the substrate topology initiates from the first ordered rack and ends on the last (***line 6***). In case that the whole SFC cannot be placed within a single rack, the *min-cut* algorithm is applied (***line 39***) to partition the request into segments, striving to minimize the amount of generated inter-rack traffic via the coordinated VNF and link assignment.

The aspect that we investigate appears during the placement of VNFs within a certain rack. Let $V' \subset V$ be the set of servers within this rack. Then, for a specific VNF $i$ of a given SFC, the corresponding $s_u^i$ values are computed in accordance to the utilized metric (***line 11***). The most suitable server for accommodating VNF $i$ is expressed by $u = argmin_{u \in V'}(s_u^i)$ or $u = argmax_{u \in V'}(s_u^i)$, depending on the metric (***line 12***). Thus, the substance of each metric lies on the way they sort the servers of each rack. Nevertheless, if the VNF can be placed in the same server as its predecessor, this placement is prioritized since, in this way, the generated traffic within the substrate network is reduced (***line 13***). Regarding the respective flow mappings, for adjacent VNFs within the same rack the mapping possibilities are limited, since there are unique paths that can establish their connection. In the case where such VNFs are accommodated in servers within different racks, though, the algorithm assigns the corresponding flow to inter-rack links that are the least loaded.

Fig. 3.3 exemplifies the logic behind the proposed algorithm, in an attempt to map an SFC to

**Algorithm 1** Heuristic for VNF placement

```
1: **Input**: $V, racks, V_F, E_F$
2: **Output**: mapping, rejected_service
3: $rejected\_service := True$
4: $N := |V_F|$
5: $s\_racks = sort(racks)$
6: for $R \in s\_racks$ do
7:     $prevServer :=$ "$None$"
8:     $placed\_vnfs := 0$
9:     for $i \in V_F$ do
10:        $placed\_vnf := False$
11:        compute $s_u^i, \forall u \in V_R$
12:        $sorted\_V_R = argsort_u(s)$
13:        if $prevServer \in sorted\_V_R$ and $i$ fits $prevServer$ then
14:            update $mapping$
15:            $placed\_vnf := True$
16:            $placed\_vnfs += 1$
17:            if $placed\_vnfs == N$ then
18:                $rejected\_service := False$
19:        else
20:            for $u \in sorted\_V_R$ do
21:                if $i$ fits $u$ then
22:                    update $mapping, prevServer$
23:                    $placed\_vnf := True$
24:                    $placed\_vnfs += 1$
25:                    if $placed\_vnfs == N$ then
26:                        $rejected\_service := False$
27:                        break
28:                else
29:                    continue
30:        if $placed\_vnf$ and $rejected\_service$ then
31:            continue
32:        else
33:            break
34:    if $rejected\_service$ then
35:        continue
36:    else
37:        break
38: if $rejected\_service$ and $|V_F| > 1$ then
39:    $V_{F_1}, V_{F_2} = minCut(V_F, E_F)$
40:    $mapping1, rejected1 = vnPlace(V, racks, V_{F_1}, E_F)$
41:    $mapping2, rejected2 = vnPlace(V, racks, V_{F_2}, E_F)$
42:    if $rejected1$ or $rejected2$ then
43:        return $\{\}, True$
44:    else
45:        return $mapping1 \cup mapping2, False$
46: else if $rejected\_service$ and $|V_F| = 1$ then
47:    return $\{\}, True$
48: else
49:    return $mapping, False$
```

a datacenter rack. In this example, we consider the embedding of an SFC (composed of three 2-dimensional VNFs) to a specific rack that consists of two servers. $d_r(i)$- and $c_r(u)$-values are used for the purposes described in Section 3.4.1. The heuristic seeks to map the VNFs sequentially, *i.e.,* it will commence with the mapping of *VNF 1* and terminate with *VNF 3*. For the sake of simplicity, let us assume that the heuristic incorporates the *Manhattan* metric. Thus, we compute the following:

$$s_1^1 = |0.45 - 0.20| + |0.75 - 0.40| = 0.60$$
$$s_2^1 = |0.80 - 0.20| + |0.75 - 0.40| = 0.95$$

and we derive that *VNF 1* will be assigned to *Server 1*. In more detail, this decision is made based on the following two conditions: (i) *Server 1* has sufficient capacity to accommodate *VNF 1* and (ii) $s_1^1 = 0.60 < 0.95 = s_2^1$ (*i.e.,* assigning *VNF 1* to *Server 1* is more efficient than assigning it to *Server 2*). Subsequently $c_r(u)$-values are recomputed, since the available resources of *Server 1* have changed. Ideally, *VNF 2* would also be placed on *Server 1*; however, it is apparent that it does not fit due to lack of available CPU resources. Thereby, *VNF 2* is mapped onto *Server 2* and $c_r(u)$-values are recomputed. We note that although *VNF 3* fits *better* in *Server 1* (*i.e.,* $s_1^3 = 0.10 < 0.25 = s_2^3$, considering the updated $c_r(u)$-values), it will be placed on *Server 2*, where its predecessor has been assigned in order to avoid the generation of additional intra-rack traffic.

This example demonstrates a critical aspect of the problem concerned. That is, striving for a balanced resource consumption may lead to larger traffic volumes and, thus, the inherent capability of an algorithm to skew in favour of the former against the latter (or the opposite) will significantly affect the overall SFCE efficiency.

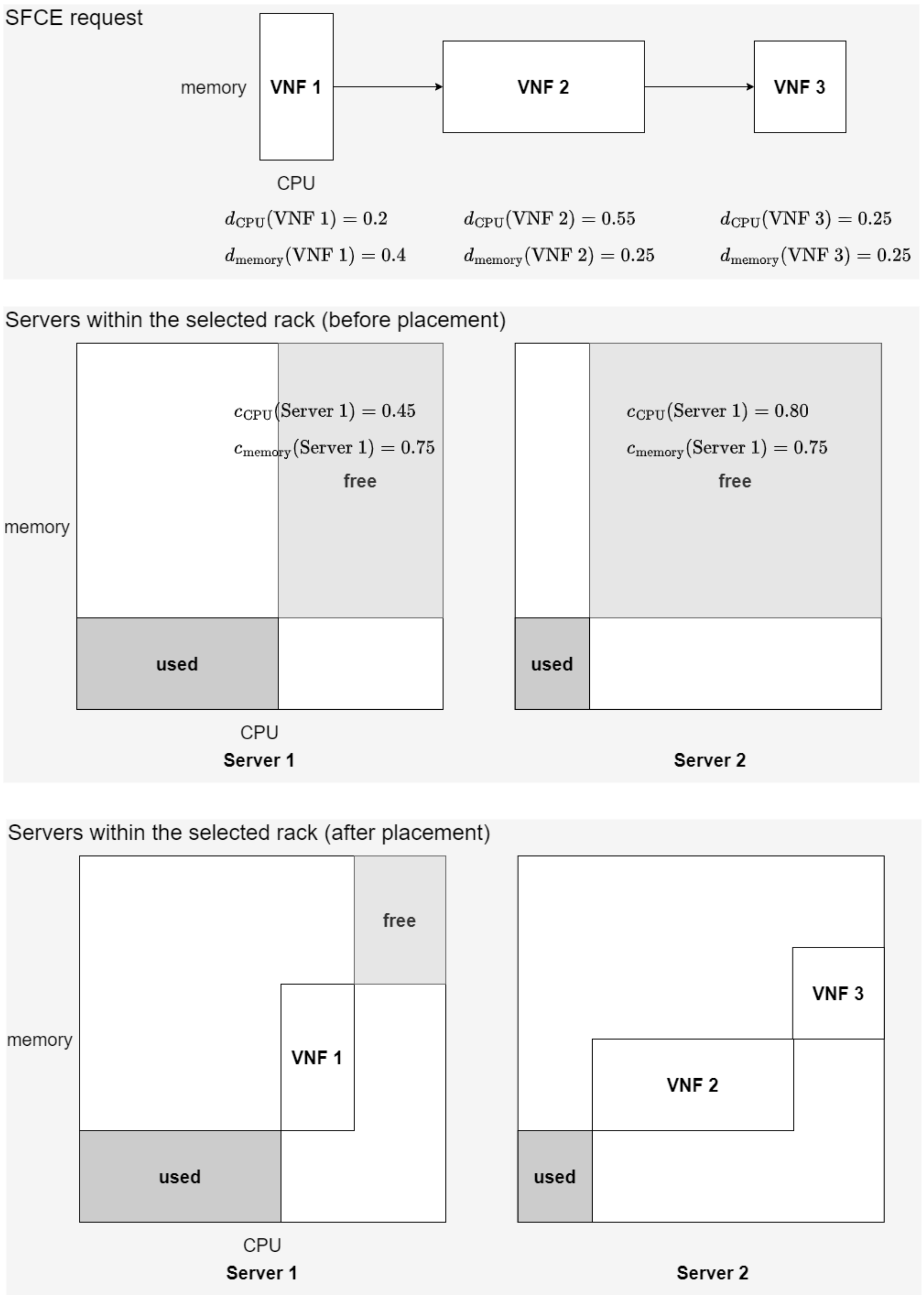

SFCE request
memory
VNF 1
VNF 2
VNF 3
CPU
$d_{\text{CPU}}(\text{VNF 1}) = 0.2$
$d_{\text{memory}}(\text{VNF 1}) = 0.4$
$d_{\text{CPU}}(\text{VNF 2}) = 0.55$
$d_{\text{memory}}(\text{VNF 2}) = 0.25$
$d_{\text{CPU}}(\text{VNF 3}) = 0.25$
$d_{\text{memory}}(\text{VNF 3}) = 0.25$
Servers within the selected rack (before placement)
$c_{\text{CPU}}(\text{Server 1}) = 0.45$
$c_{\text{memory}}(\text{Server 1}) = 0.75$
free
$c_{\text{CPU}}(\text{Server 1}) = 0.80$
$c_{\text{memory}}(\text{Server 1}) = 0.75$
free
memory
used
used
CPU
Server 1
Server 2
Servers within the selected rack (after placement)
free
memory
VNF 1
VNF 3
VNF 2
used
used
CPU
Server 1
Server 2

Figure 3.3: SFCE example with the multi-dimensional heuristic using the *Manhattan* metric.

### 3.4.4 Baseline heuristic

Moreover, we have developed an additional heuristic called *heuristic-cpu*. This heuristic acts as a reference point, allowing us to measure the potential benefits of SFCE when considering multiple resource dimensions. Specifically, this heuristic is similar to the one described in Section 3.4.3. However, instead of utilizing any of the mapping efficiency metrics outlined in the preceding sections, it prioritizes the servers within a rack based on their CPU capacity, ranking them in descending order.

### 3.4.5 Greedy VNF bundling

We devise a VNF-graph pre-processing method, namely *Greedy VNF Bundling* (*GVB*), which seeks to generate bundles out of VNFs, while maintaining the same resource requirements with the sum of individual VNFs that comprise each bundle. The primary aim of GVB is to create VNF bundles with more balanced resource demands (across the resource dimensions), thereby, augmenting the underlying SFCE method in VNF placement. The main intuition behind this approach is that bundling VNFs with resource intensity in different dimensions (*e.g.,* a CPU-intensive VNF with a memory-intensive one) can generate more balanced VNF bundles. One restriction here is that bundling should be applied only to consecutive VNFs (in the service chain) in order to enforce correctness in the processing of traffic.

A key-feature of GVB is the way that it groups adjacent VNFs. More specifically, it commences with an empty bundle, and places the first VNF of the chain within it. Subsequently, it assigns the following VNF in the same bundle, if the sum of the respective VNF resource requirements results in a more resource-balanced (bundled) VNF, provided that its requirements do not exceed the capacity of a server; otherwise, it opens up a new bundle and repeats the same procedure. The level of resource balance is quantified using Eq. (3.4), where $\mathbf{c}(u) = \vec{1}$, and $\mathbf{d}(i)$ expresses the resource requirements of the $i^{th}$ bundle.

To exemplify GVB, Fig. 3.4 illustrates a service graph comprising four VNFs, with their respective resource demand $\mathbf{d}(i)$ vectors (*i.e.,* normalized CPU, memory, and storage demands,

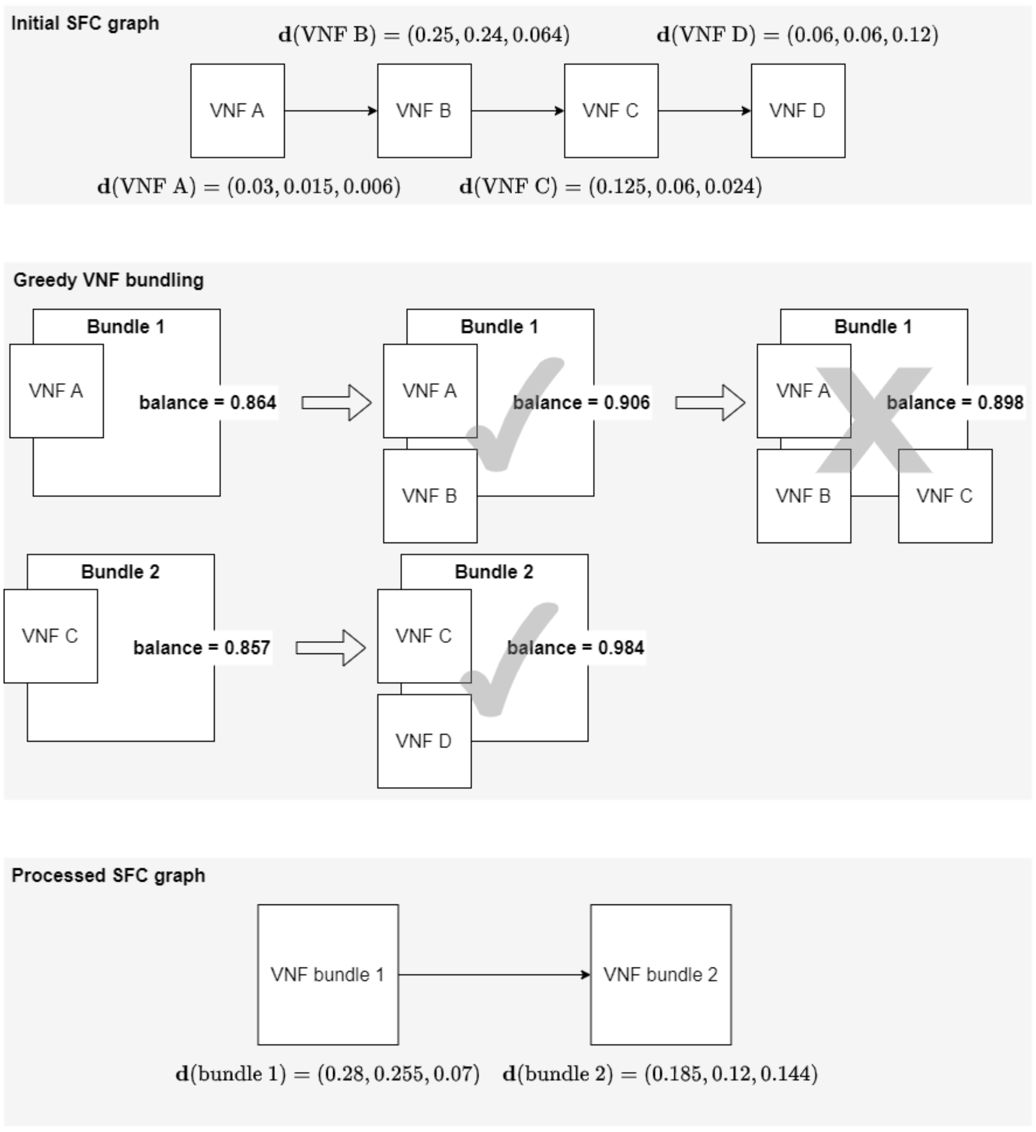

Initial SFC graph
$\mathbf{d}(\text{VNF B}) = (0.25, 0.24, 0.064)$
$\mathbf{d}(\text{VNF D}) = (0.06, 0.06, 0.12)$
VNF A
VNF B
VNF C
VNF D
$\mathbf{d}(\text{VNF A}) = (0.03, 0.015, 0.006)$
$\mathbf{d}(\text{VNF C}) = (0.125, 0.06, 0.024)$
Greedy VNF bundling
Bundle 1
VNF A
balance = 0.864
Bundle 1
VNF A
VNF B
balance = 0.906
Bundle 1
VNF A
VNF B
VNF C
balance = 0.898
Bundle 2
VNF C
balance = 0.857
Bundle 2
VNF C
VNF D
balance = 0.984
Processed SFC graph
VNF bundle 1
VNF bundle 2
$\mathbf{d}(\text{bundle 1}) = (0.28, 0.255, 0.07)$
$\mathbf{d}(\text{bundle 2}) = (0.185, 0.12, 0.144)$

Figure 3.4: The *Greedy VNF Bundling* (GVB) service graph pre-processing method.

respectively). For instance, $\mathbf{d}(A) = [.03, .015, .006]$ indicates that *VNF A* requires 3% of a server's CPU, 1.5% of its memory, and 0.6% of its storage capacity. According to the GVB procedure, an empty bundle (*i.e., Bundle 1*) is spawned, and *VNF A* is assigned to it. Eq. (3.4) computes the level of resource-balance of *Bundle 1*, as follows:

$$balance(Bundle1) = \cos\theta = \frac{\vec{1} \cdot [.03, .015, .006]}{||\vec{1}|| \cdot ||[.03, .015, .006]||} = .864$$

Subsequently, we examine whether *VNF B* should be placed into *Bundle 1.* This will result in

*Bundle 1* requiring

$$[.03 + .25, .015 + .24, .006 + .064] = [.28, .255, .07]$$

resources, and thus:

$$balance(Bundle1) = \cos\theta = \frac{\vec{1} \cdot [.28, .255, .07]}{||\vec{1}|| \cdot ||[.28, .255, .07]||} = .906$$

Since the generated bundled VNF is more balanced, we assign *VNF B* to *Bundle 1.* Next, we attempt to assign *VNF C* to *Bundle 1*, also. In this case, we obtain:

$$balance(Bundle1) = \cos\theta = \frac{\vec{1} \cdot [.405, .315, .094]}{||\vec{1}|| \cdot ||[.405, .315, .094]||} = .898$$

which implies a degraded level of resource balance. Hence, we do not assign *VNF C* to *Bundle 1*; instead, a new bundle (*i.e., Bundle 2*) is spawned. Following a similar approach, we obtain that *Bundle 2* comprises *VNF C* and *VNF D.* Finally, this pre-processing phase results in a new service graph (depicted at the bottom of Fig. 3.4), which consists of two VNF bundles connected with a virtual link (*i.e.,* the edge between *VNF B* and *VNF C*).

## 3.5 Evaluation Results

### 3.5.1 Evaluation setup

All of the algorithms assessed are designed to embed SFC requests within a datacenter that features a two-tier hierarchical structure. In the simulation, the datacenter comprises 200 servers, grouped into 10 racks. The 10 top-of-the-rack (ToR) switches interface through the core layer of the topology, which comprises 5 core switches. Each individual server offers a computing capacity of 64 vCPUs, along with 512 GB of primary memory, and 8 TB of storage space. Additionally, every server is linked to its respective ToR switch at a rate of 4 Gbps. In contrast, the connections between the ToR and the core switches operate at 16 Gbps. These specifications can be viewed in Table 3.1.

Leading cloud service providers, such as Microsoft [54], Amazon [55], Google [56], and IBM [57], facilitate the deployment of *compute-*, *memory-*, and *storage*-optimized VMs, to name a few. Such offerings underscore the diverse resource needs of tasks carried out in cloud settings, as well as the adaptability the cloud affords in meeting these varied demands. Drawing from these insights, we categorize VNFs into three main types: *compute-*, *memory-*, and *storage*-optimized VNFs. Depending on the type, a VNF instance could require 2, 4, 8, 16, or 32 vCPUs. However, the requirements for other resources will vary based on the specific VNF type. To elaborate:

**Compute-optimized** VNF instances are specified by a constant memory-to-vCPU ratio of 4, as well as a constant storage-to-vCPU ratio of 25.

**Memory-optimized** VNF instances are associated with a constant memory-to-vCPU ratio of 8 and a constant storage-to-vCPU- ratio of 32.5.

**Storage-optimized** VNF instances have a constant memory-to-vCPU ratio of 8 and a constant storage-to-vCPU ratio of 232.5.

Therefore, if a VNF requires 8 vCPUs and is classified as a memory-optimized instance, the required memory and storage capacities will be 64GB and 260GB, respectively. Apparently, these VNF instances yield a *positive correlation* between the various resource dimensions (pair-wise), irrespective of their type. Although this correlation may influence the behavior of our algorithms, such instances are deemed more realistic.

Moreover, the incoming traffic for an SFC is assigned a random value between $[10, 100]$ Mbps. In contrast, the ratio of inbound to outbound traffic for each VNF within the service chain is set to a random value within the range of $[0.5, 1.5]$. This particular ratio sheds light on how VNFs might influence the magnitude of the outbound traffic in relation to the inbound traffic. Such variations can arise because certain VNFs, like those responsible for packet inspection, redundancy removal, or caching (among other tasks) may suppress or amplify the inbound traffic [31].

In terms of incoming SFC requests, we consider the more realistic case of expiring SFCs with a lifetime of 3 to 9 time intervals. Such SFC requests arrive at the datacenters with a Poisson distribution (30 on average per time interval). Furthermore, each SFC comprises 3 to 10 VNFs (cf. Table 3.3).

Our simulations are conducted on a server with 8 CPU cores at 2.1 GHz and 8 GB of RAM, using the Linux-based Ubuntu 16.04 (LTS) operating system. The simulation environment is implemented in Python, while for the MILP we rely on the Gurobi solver [58].

For the comparison between the SFCE methods, we use the following metrics:

**Request acceptance rate**, which is computed as the ratio of the successfully embedded SFCs over the total number of incoming SFC requests.

**Resource utilization**, which corresponds to the sum of the physical resources (*i.e.,* CPU, memory, storage) that have been allocated for the VNFs (and are hence monetized by the InP).

**Resource skewness**, which is computed according to Eq. (3.6) (cf. Section 3.4.1).

**Intra- and inter-rack traffic**, which is the mean volume of traffic observed in the respective links in various snapshots of the simulated network infrastructure.

### 3.5.2 Evaluation of metrics

In this section, we explore the effectiveness of the two multi-dimensional mapping metrics, specifically the *L2 norm-based greedy* and *cosine similarity*, as they've been integrated into our suggested heuristic. As demonstrated in Fig. 3.5, the heuristics utilizing these metrics prove to be more proficient than the heuristic-cpu. Notably, the *heuristic-L2* displays a higher acceptance rate than the *heuristic-cos*, but both clearly surpass the performance of the third algorithm. Bearing in mind that a 1% difference in the acceptance rate at the end of the experiment corresponds to the rejection of more than 70 SFCs, the improvement by the proposed heuristics is deemed considerable.

Table 3.1: Datacenter Model Parameters

| | |
|---|---|
| Number of Core Switches | 5 |
| Number of Racks | 10 |
| Number of Servers per Rack | 20 |
| Capacity of server vCPUs | 64 |
| Capacity of server memory | 512GB |
| Capacity of server storage | 8TB |
| ToR-to-Server link capacity | 4Gbps |
| Inter-rack link capacity | 16Gbps |

Table 3.2: SFC Request Model Parameters

| | |
|---|---|
| Inbound SFC traffic | uniform distr. [10,100] |
| Outbound-to-inbound traffic ratio | uniform distr. [0.5,1.5] |
| Splittable flow | Yes |

Table 3.3: Evaluation Environment Parameters

| | |
|---|---|
| Number of SFCs per time interval | Poisson(30) |
| Number of VNFs per SFC | uniform distr. [3,10] |
| SFC type | expiring |
| SFC running duration | random [3,9] intervals |

To delve deeper into embedding efficiency, we also assess the resource utilization achieved by the three aforementioned methods. In doing so, we present the percentage difference in resource utilization between the heuristic-L2 and heuristic-cos (calculated as a moving average) relative to that of the heuristic-cpu. As indicated in Fig. 3.6, the proportion of surplus CPU allocated to SFCs by the multi-dimensional heuristics is either equivalent or lesser than the corresponding percentages related to the other resource dimensions (see Figs. 3.7, 3.8). This holds true for both heuristic-L2 and heuristic-cos. Such a result aligns with expectations, given that the heuristic-cpu prioritizes the CPU resource dimension, thereby allocating it to VNFs more judiciously than it does for memory and storage. Nonetheless, when considering the allocation efficiency across all resources, there are notable improvements: a minimum gain of 4% for the heuristic-cos and at least 7% for the heuristic-L2.

To further comprehend the advantages of the multi-dimensional heuristics, we explore how resource balance plays a pivotal role in their efficiency. As illustrated in Fig. 3.9, the introduced algorithms demonstrate a more balanced consumption of the various resources compared to the heuristic-cpu. When measuring this balance through the *resource skewness* metric, the two

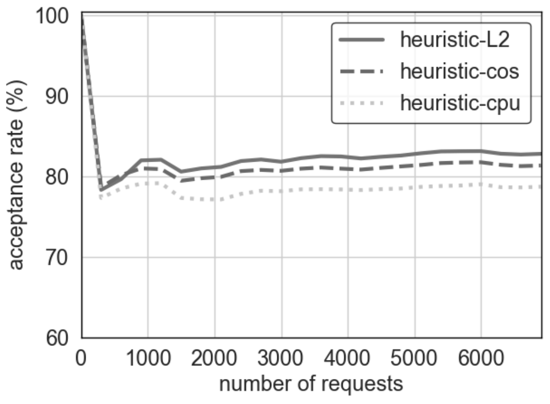


Figure 3.5: Request acceptance rate.

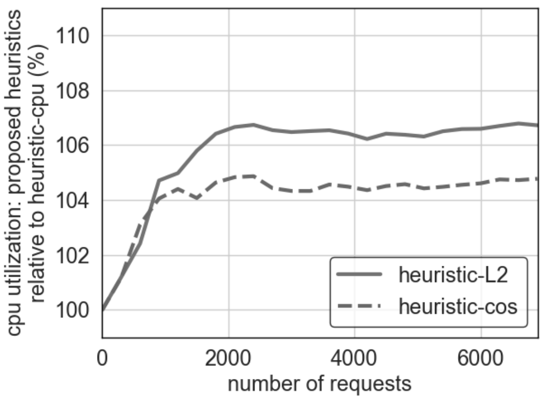


Figure 3.6: CPU utilization of the multi-dimensional heuristics, in relation to the single-dimensional.

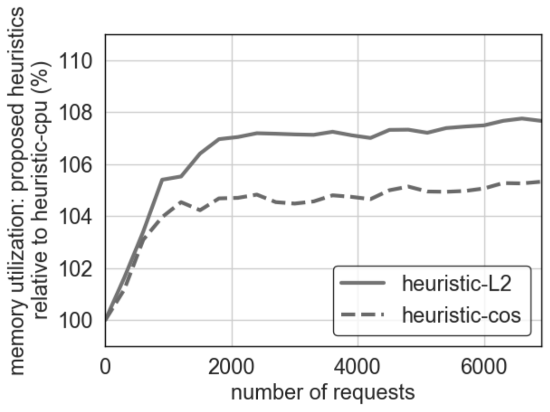


Figure 3.7: Memory utilization of the multi-dimensional heuristics, in relation to the single-dimensional.

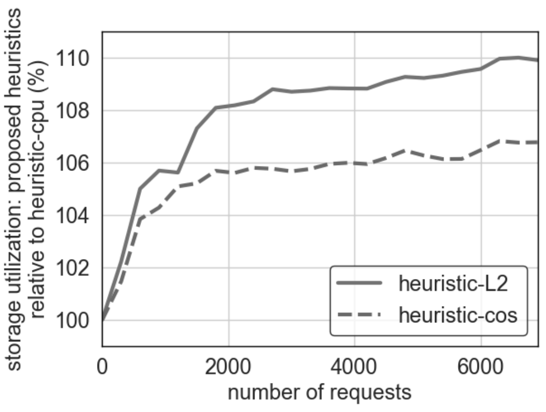


Figure 3.8: Storage utilization of the multi-dimensional heuristics, in relation to the single-dimensional.

heuristics produce server states that are nearly 8% more balanced on average. Moreover, the algorithm utilizing the *cosine similarity* metric exhibits the most favorable skewness, as its skewness values typically fall below those of the heuristic-L2. Yet, it would be premature to conclude that the enhanced resource utilization is purely a result of skewness (*i.e.,* harmonized consumption of server resources). If this were solely the case, the heuristic-cos would offer similar, if not superior, efficiency compared to the heuristic-L2. In contrast, our earlier findings suggest otherwise.

This finding essentially raises the need for a deeper investigation of the way that the two

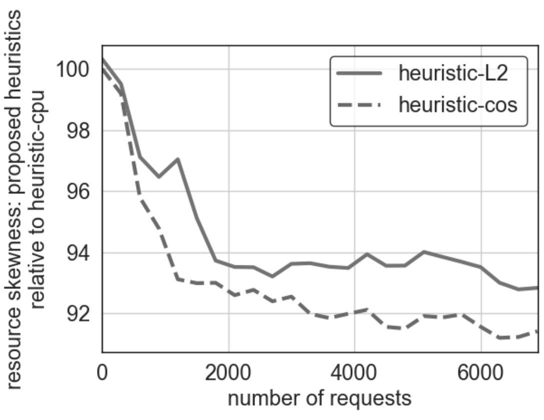


Figure 3.9: Resource skewness of the multi-dimensional heuristics, in relation to the single-dimensional (lower is better).

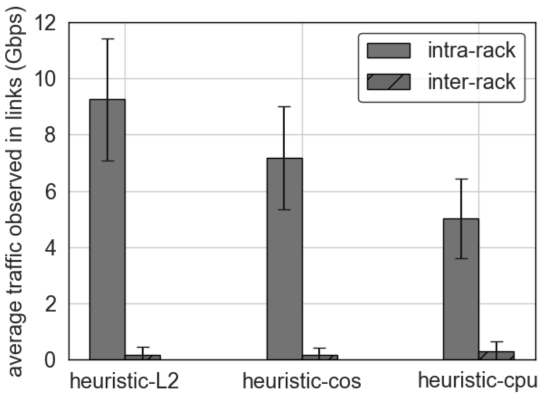


Figure 3.10: Intra-rack and inter-rack traffic distribution.

metrics influence the embedding decisions. Recall that the *cosine similarity* metric ranks servers according to (3.4). As such, a previously unused server may be selected for the placement of a VNF, although this VNF may fit into a previously allocated server within the same rack. This can, in turn, lead to the mapping of subsequent VNFs onto the same server (since it currently hosts only one VNF), although (according to Algorithm 1) there will be no consideration of the degree of alignment between the VNF resource requirements and the available resources of the server.

Consequently, the heuristic-cos utilizes the *cosine similarity* less frequently in the computation of VNF-to-server mappings, which eventually affects its efficiency. On the other hand, the heuristic-L2 tends to map a VNF to the server that minimizes (3.1) (*i.e.,* its embedding approach is dictated by the *Best Fit* principle). As such, subsequent VNFs are less likely to be placed onto the same server, due to lack of available resources. Thereby, the *L2 norm-based greedy* metric is used more frequently for the computation of the additional VNF-to-server assignments.

Along these lines, we reach the following observations: (i) the heuristic-L2 yields a higher degree of server consolidation compared to the heuristic-cos, and (ii) the effectiveness of a multi-dimensional mapping efficiency metric is highly dependent on the embedding heuristic. These observations are essentially reflected by our resource allocation and skewness results.

The *cosine similarity*, in particular, has been proved a more efficient metric compared to the *L2 norm-based greedy* [59]. However, this gain of the *cosine similarity* is not translated into better embedding efficiency, since the corresponding metric is less frequently utilized by the embedding heuristic.

Lastly, Fig. 3.10 illustrates the average intra- and inter-rack traffic generated on the datacenter. According to this plot, all algorithms achieve a negligible degree of inter-rack traffic, with the heuristic-cpu exhibiting higher variation from the average value. This is attributed to the more frequent *min-cut* triggering, *i.e.,* the algorithmic element that enables the partitioning of an SFC across multiple racks. We also observe a higher volume of intra-rack traffic for the heuristic-L2. This stems from the following observations: (i) the heuristic-L2 admits a larger number of incoming SFCs, and (ii) it exhibits a tendency for SFC partitioning across multiple servers, as explained in our previous discussion about the *L2 norm-based greedy* metric.

### 3.5.3 Comparison of MILP and heuristic-L2

As detailed in Section 3.4.2, we integrate the *Manhattan distance* into an MILP to penalize suboptimal assignments, all while maintaining the linearity of the exact method. Subsequently, we compare the derived algorithm (*i.e., MILP-manhattan*) with the heuristic-L2, which stood out from the previous comparison. To this end, we assess the margin between the heuristic algorithm and the optimal solutions computed by the *MILP-manhattan.*

According to Fig. 3.11, the two algorithms manage to embed roughly the same number of SFCs. Their marginal difference of $\sim 1\%$ (that unexpectedly favors the heuristic-L2) is further investigated via additional micro-benchmarks. More precisely, an analysis of the rejections (Fig. 3.12) uncovers the limited computational capacity of the heuristic for embedding long ($\geq 8$ VNFs) SFCs compared to the MILP; instead, the heuristic utilizes the available resources for embedding more short SFCs ($\leq 5$ VNFs). This computational limitation of the heuristic-L2 can be mainly attributed to its poor *search space exploration* capability (compared to the *MILP-manhattan*), since the exercised *min-cut* policy reduces the VNF-to-server allocation combinations.

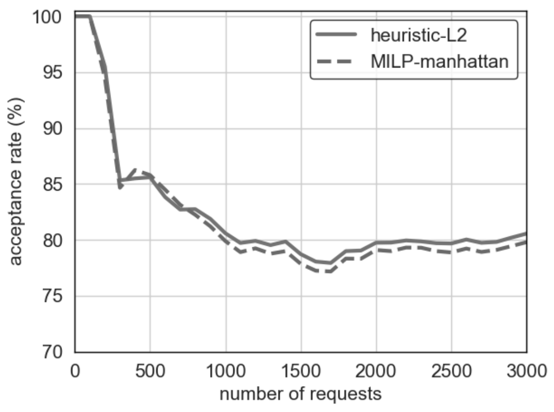


Figure 3.11: Request acceptance rate.

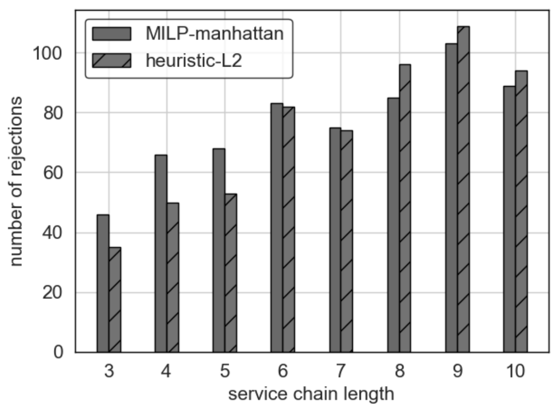


Figure 3.12: Number of SFC rejections for the different service chain lengths.

After approximately 3,000 SFCE requests, we observe the relative utilization for each resource dimension, illustrated in Fig. 3.13. It is apparent that storage (which is the most abundant resource type) has the most profound impact on the overall resource utilization of the heuristic-L2. As such, we conclude that the proposed heuristic exhibits only a minimal (1-2%) margin from the MILP in terms of maximizing resource utilization. This, combined with the improved resource skewness (cf. Fig. 3.14), empower the heuristic-L2 to achieve high efficiency on par with the *MILP-manhattan*.

However, there is a noticeable disparity between the two methods in terms of consolidation, with the *MILP-manhattan* having an edge. This is suggested by the corresponding intra-rack traffic levels displayed in Fig. 3.15. Nevertheless, MILP exhibits a significant drawback, as its solver run-time grows exponentially with the problem size. For instance, according to Fig. 3.16, the computation of the embedding for an SFC with 10 VNFs requires 6 seconds on average, whereas the heuristic's run-time is in the order of milliseconds across all SFC lengths tested here. The scalability limitation of the MILP could be potentially addressed by employing relaxation and rounding techniques, similar to our previous work in [31]. However, the resulting linear program (LP) is expected to yield a degree of sub-optimality (compared to the MILP), whereas its solver run-time would still be considerably higher than the heuristic.

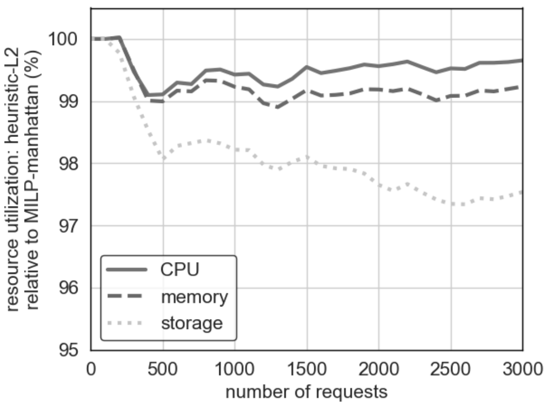


Figure 3.13: Resource utilization of the heuristic-L2, in relation to the MILP-manhattan.

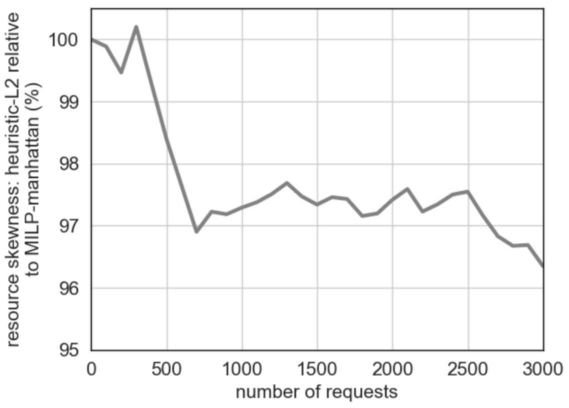


Figure 3.14: Resource skewness of the heuristic-L2, in relation to the MILP-manhattan.

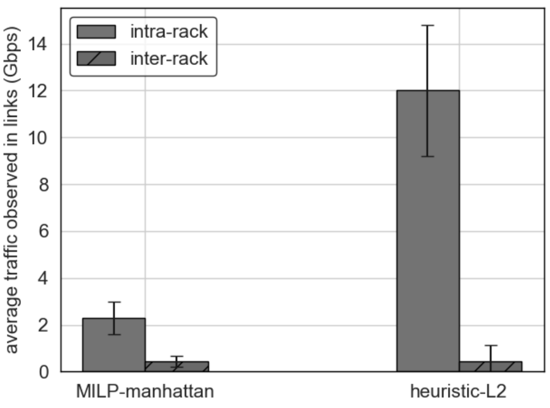


Figure 3.15: Intra-rack and inter-rack traffic distribution.

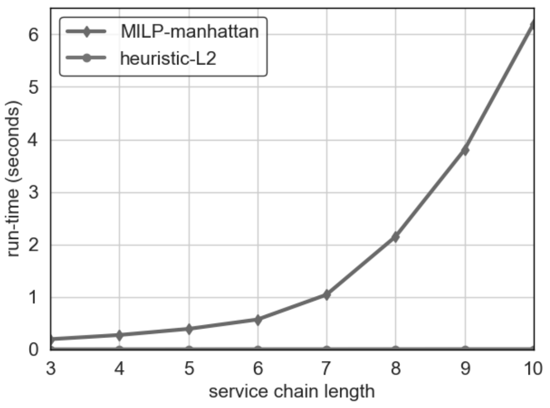


Figure 3.16: Average solver run-time for the different service chain lengths.

## 3.5.4 Comparison of GVB-heuristic-L2 and heuristic-L2

We hereby investigate the potential gains from the GVB pre-processing method, which generates VNF bundles in order to augment VNF embedding. To assess the efficiency of GVB, we couple it with the heuristic-L2, which stands out as the most prominent embedding method (in the following, we term this combination as *GVB-heuristic-L2*). In particular, GVB processes the initial service graph generating a potentially shorter graph with bundled VNFs, which is, in turn, conveyed to the heuristic-L2 for optimized embedding. Since VNF bundles are associated with higher resource requirements (compared to the VNFs of the initial service graph), this reduces the search space for VNF embedding optimization. Consequently, if the embedding of

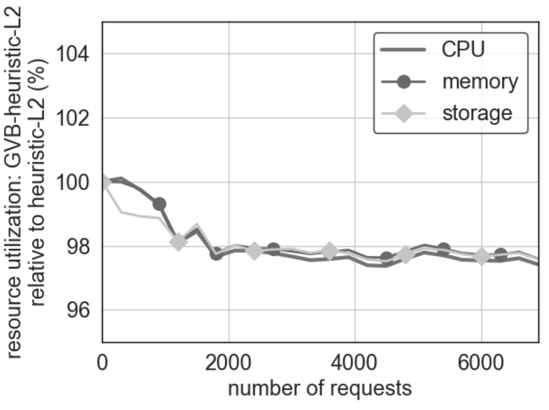


Figure 3.17: Resource utilization of the GVB-heuristic-L2, in relation to the heuristic-L2.

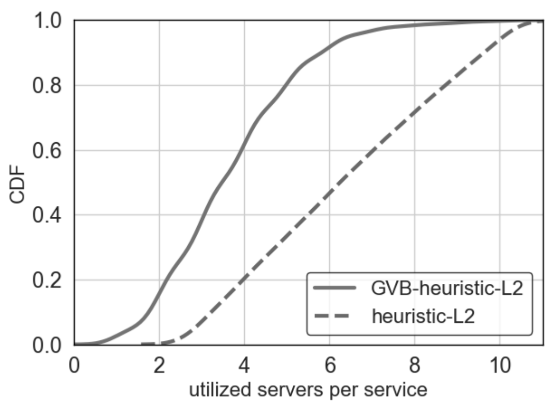


Figure 3.18: CDF of the number of servers utilized per network service.

a bundled VNF is not possible, instead of the embedding request rejection, we perform another embedding attempt using the initial (non-bundled) service graph. In such case, we assign *zero* to a *GVB-success* binary variable. We also assign *zero* to the same variable, whenever the number of generated bundles equals the number of initial VNFs (which might occur if, for instance, all VNFs of a service chain are CPU-intensive). Otherwise (*i.e.,* whenever GVB-heuristic-L2 successfully embeds bundled VNFs), we assign the value of *one* to this variable.

According to Fig. 3.17, the assignment of CPU, memory, and storage of the GVB-heuristic-L2, in relation to the respective allocations of the heuristic-L2, is approximately 2% less across all three resource dimensions. This implies slightly lower profits for an InP; however, the VNF bundling method appears to yield higher embedding efficiency than heuristic-L2, as indicated in Fig. 3.18. More precisely, 90% of the services embedded with the GVB-heuristic-L2 are assigned to 5 servers at most, whereas the respective number of servers for heuristic-L2 is 9. Although these servers usually belong to the same rack, the higher degree of VNF co-location achieved by GVB-heuristic-L2 reduces the volume of generated traffic.

To shed more light on the behavior of the bundling algorithm, we examine its success rate. In Fig. 3.19, a continuous line illustrates the product of the *GVB-success* variable with the average CPU utilization of the datacenter, whereas the dashed line merely represents the latter. These curves intersect whenever a bundled VNF placement occurs, *i.e., GVB-success=1.*

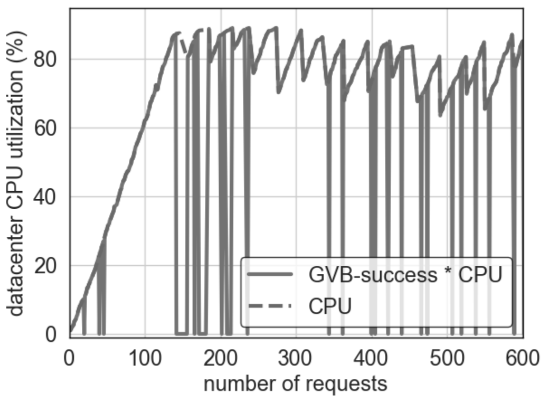


Figure 3.19: GVB triggering frequency.

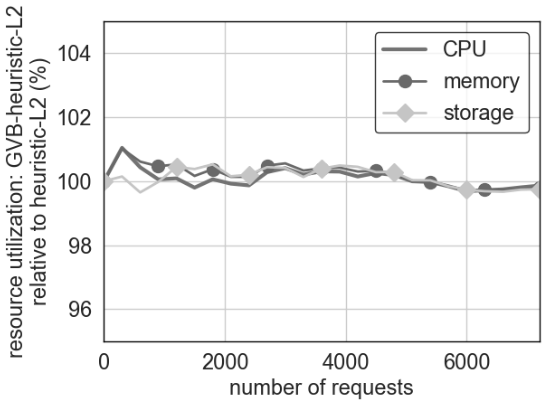


Figure 3.20: Resource utilization of the GVB-heuristic-L2, in relation to the heuristic-L2 (threshold-based variant).

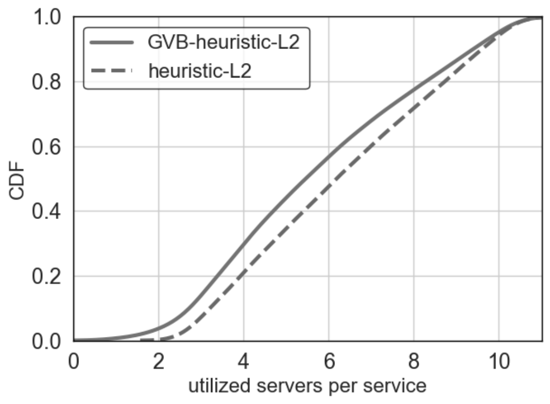


Figure 3.21: CDF of the number of servers utilized per network service (threshold-based variant).

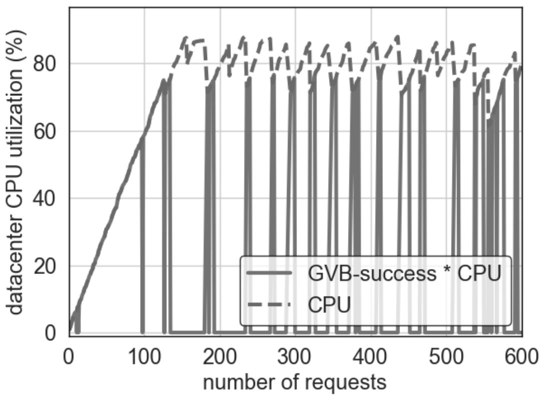


Figure 3.22: GVB triggering frequency (threshold-based variant).

Unexpectedly, the GVB-heuristic-L2 yields *successes* even at very high utilization levels (*e.g.,* above 80%), which indicates that GVB is triggered very frequently.

The insights gained from the behavior of GVB-heuristic-L2 lead to a critical observation, which stems from an intuitive principle of bin-packing, *i.e.,* having many and small items, instead of fewer and larger, when closing up the bins (*i.e.,* when bins are almost full), is more effective. Therefore, we evaluate a variant of GVB-heuristic-L2, whose main difference from the initial bundling scheme is that it is not executed when the infrastructure exceeds a certain utilization level. In our simulations, we set this threshold to 75% of the mean DC CPU utilization. As such, this GVB variant is triggered only for low and medium utilization levels. Instead, it reduces to the basic heuristic-L2 for the embedding of the initial VNFs (in analogy to many and smaller items), when the infrastructure utilization is high (analogous to bins ready to close up). With this behavior, GVB lays the foundation for a balanced resource consumption within servers.

This adjustment in the behavior of GVB yields even higher resource efficiency, since it leads to (i) equivalent resource allocations with the heuristic-L2 (Fig. 3.20) and (ii) slightly improved VNF co-location efficiency, according to Fig. 3.21. Fig. 3.22 corroborates that GVB is triggered (frequently) only at low and medium utilization levels.

Consequently, the GVB-heuristic-L2 effectively couples the *cosine-similarity* (according to the bundling logic described in Section 3.4.5) with the *L2 norm-based greedy* (since the heuristic-L2 is employed to map the service graph with the bundled VNFs). Thereby, the GVB-heuristic-L2 promotes the synergy of the two metrics, whose gains are illustrated in Figs. 3.20 and 3.21. Additional gains could be achieved by coupling the proposed bundling scheme with an SFCE MILP, as well. In particular, the smaller number of VNFs in the service graph (as the outcome of VNF bundling) is expected to reduce the MILP solver run-time, which is illustrated in Fig. 3.16.

## 3.6 Related Work

Throughout this section, we discuss related work on the following two areas: (i) SFCE and (ii) multi-dimensional VM assignment.

### 3.6.1 Service Function Chain Embedding

*Benson et al.* propose CloudNaaS [60], a system that allows the deployment of cloud applications offering features such as virtual network isolation, service differentiation and flexible deployment of middleboxes. The network controller of CloudNaaS is responsible, among others, for the assignment of VMs (that compose the cloud application) to physical servers considering their communication dependencies and requirements. Authors employ a heuristic approach for tackling this problem, with its ultimate goal being to minimize the amount of flow mapped on the network links. Another research work that considers the problem of SFCE is Stratos [61]. In particular, Stratos comprises a framework for a multitude of NFV orchestration functions, such as service chaining and resource provisioning. The resource provisioning aspect of Stratos, which is the most relevant to our work, mainly deals with flow distribution, *i.e.,* the assignment of flows to the various VNF instances. This problem is formulated as a linear program.

Nestor [31] and DistNSE [62] tackle the multi-provider SFCE problem, taking into account the privacy restrictions imposed by the different participating providers. Both Nestor and DistNSE decompose this into two sub-prolems: (i) service chain partitioning (among providers), and (ii) mapping of each chain segment onto the respective provider's domain. While Nestor relies on a centralized broker for service chain partitioning, DistNSE handles this in a distributed manner, using an embedding protocol (which essentially obviates the need for any centralized orchestrator). The intra-provider mapping of both Nestor and DistNSE (which is the particular sub-problem of relevance to our work) is restricted to a single dimension for all resource types (*i.e.,* CPU demands for VNFs and bandwidth demands for links).

In [63], *Fu et al.* present a deep reinforcement learning approach for the embedding of network services. As implied in the paper, the inherent dynamic nature of Internet of Things should be supported by SFCs being embedded by algorithms that can autonomously adjust their decisions based on past knowledge.

The SFCE problem has also been studied in the context of cellular network slicing. The aim here is to optimize the placement of VNFs that implement certain elements of a virtualized cellular network, such as the eNodeB (eNB), the Serving Gateway (S-GW), the Packet Data Network Gateway (P-GW), and the Mobility Management Entity (MME). In particular, authors in [46] propose an exact method for the embedding of such VNF-graphs onto a virtualized cellular core. Besides capacity constraints, the proposed method optimizes the placement of VNFs, taking into account delay budgets between VNFs, as mandated by 3GPP. *Papagianni et al.*

follow a different approach to this SFCE problem, as they promote the sharing of VNFs among multiple service chains in a network slice, as means to reduce the deployed VNF instances and, consequently, the provisioning and management cost of the virtualized cellular network [3]. To this end, the authors propose a MILP formulation for SFCE with shared VNFs.

Even though all of the above-mentioned studies shed light on critical aspects of SFCE, the fact that they are confined in single-dimensional nodes (*i.e.,* CPU demand) restricts their applicability to SFCs with diverse requirements across multiple resources. Hence, we address this limitation by providing an efficient SFCE algorithm that accounts for multiple resource dimensions, eventually leading to more efficient utilization and higher revenues for InPs.

### 3.6.2 Virtual Machine Assignment

The multi-dimensional VM allocation problem is examined as a specific application of the more generic VBP problem. Within the context of mapping VMs to physical hosts, several studies (*e.g.*, [53, 64]) primarily focus on server consolidation, *i.e.,* the allocation of VMs into as few servers as possible.

*Ghodsi et al.* [65] is considered to be one of the most influential studies on the discussed topic. This study is among the first that approaches the fair allocation of multiple resources as pertaining to users with heterogeneous requirements. To this end, they propose Dominant Resource Fairness (DRF), a multi-dimensional allocation scheme that meets a multitude of preferable properties. However, the applicability of DRF can be seen as more suitable for large computer clusters, where all users have equal rights of utilizing their resources, as opposed to commercial cloud infrastructures, at which profit maximization is commonly sought.

An additional research work tackling this problem is [53]. In particular, *Xiao et al.* implement an algorithm that aims at using the least amount of hosts possible. In addition, the utilization of host resources must not exceed respective pre-determined thresholds so as to avoid VM performance degradation. In order to circumvent the difficulty of allocating resources across multiple dimensions, authors embrace the notion of resource skewness. Essentially, skewness acts as means to quantify the efficiency of server resource utilization and also trigger VM migrations that improve the overall server utilization.

Motivated from VM placement problems, the studies in [48, 50] propose new heuristics for the multi-dimensional VM consolidation and the VBP problem in general. In both studies, authors underline the difficulty of extending the objective of *best fitting* VMs to servers, when considering both of them as multi-dimensional vectors. To this end, they propose two heuristics; namely, the *Dot Product* and the *Norm-based Greedy*. The former is able to evaluate the VM requirements against the residual capacity (across multiple dimensions) of servers, while the latter utilizes a certain norm to quantify the distance between the aforementioned vectors.

Tetris [59], a cluster scheduler that matches tasks with machines according to requirements along multiple resources, is proposed to confront the drawbacks of fairness- and slot-based schedulers. Authors suggest that the former does not optimize job completion time, while the latter results in either excessive resource allocation or resource wastage. To this end, Tetris incorporates the *cosine similarity* (defined in Section 3.4.1) for calculating the alignment of tasks with servers.

The objective of our study is aligned with related work [31, 61, 60], which targets at the minimization of the generated inter-rack traffic and also ensures correctness for the embedded service chains. Since we deal with multi-dimensional virtual and substrate nodes, we further employ the efficiency metrics used in [48, 50, 59, 53], which are discussed in detail in Section 3.4.1. To the best of our knowledge, the coupling of SFCE with the multi-dimensional VM allocation problem has not been investigated in the literature. Our work essentially enables the computation of efficient SFC embeddings under a pragmatic scope, which leads to a more balanced resource consumption, eliminating resource wastage and potential revenue losses for InPs.

## 3.7 Conclusions

We conducted a comprehensive study on SFCE across multiple resource dimensions. To this end, we evaluated the effectiveness of several multi-dimensional mapping metrics initially introduced for multi-dimensional VM assignments. By incorporating these metrics, namely *L2 norm-based greedy*, *cosine similarity*, and *Manhattan distance*, into either heuristic or exact methodologies, we were able to quantify the advantages of multi-dimensional SFCE.

Based on the VNF types and their associated resource demands, along with the initial resource capacities of the servers used in our simulations (as detailed in Section 3.5.1), two observations can be made: (i) there is a positive pairwise correlation among different resource dimensions, and (ii) the CPU resource is the most sought after (this observation depends on the simulation settings, but it is on par with real-life deployments). These factors undeniably influence the efficacy of a mapping metric. Indeed, we ran tests with VNFs demanding a uniformly random resource quantity ranging from 1% to 50% of a server's initial resources across all dimensions. In this scenario, the heuristic-cos emerged as the top performer across all performance indicators. However, such parameters are not reflective of real-world situations, leading us to exclude them from our primary evaluation.

The *Manhattan distance* between the vector of residual capacities and the vector of demands, as pertaining to a server and a VNF, respectively, takes values that are tightly related to the number of different resource types, *i.e.,* resource dimensions. For $n$ dimensions, and given that all vector coordinates are normalized, the $M_u$ could theoretically lie within the range [0,n). Therefore, the $\alpha$ value is adjusted to 10, considering (i) that we are dealing with three dimensions, (ii) the domains of the other variables, and (iii) that we wanted to emphasize the MILP's capability of obtaining efficient VNF-to-server assignments. At the same time, as already discussed in Section 3.5, this inherently strengthens the algorithm's capability of consolidating the VNFs of a specific SFC. Nevertheless, there will be a pivotal value of $\alpha$ that, if exceeded, the second term of (3.16) will be dominated, resulting in inefficient SFCEs, due to unfavourably large inter-rack traffic volumes within the datacenter.

Our evaluation results corroborate the improved resource efficiency of multi-dimensional SFCE compared to a single-dimensional counterpart. The resource efficiency gains mainly stem from (i) a more balanced resource consumption, and (ii) increased VNF consolidation, which confines both intra- and inter-rack traffic in the DC. Additional insights gained from our evaluations indicate that the way resource consumption balance is achieved is crucial, since the heuristic-L2 and the heuristic-cos achieve similar resource skewness levels, yet they exhibit notable differences in the rest of the performance metrics. This brings us to a second conclusion, which is the critical impact of the embedding algorithm on the utilization of the mapping efficiency metric. In our case, this pertains to the frequency that the metric is utilized. Furthermore, we identified an insignificant gap in the optimality between the heuristic-L2 and the MILP (which is essentially outweighed by the substantial solver run-time of the latter).

Lastly, we evaluated a VNF graph pre-processing method, which strives to group VNFs into resource-balanced bundles. Our evaluation results indicate that the proposed bundling scheme, coupled with the heuristic-L2 (*i.e.,* GVB-heuristic-L2), can increase resource utilization while achieving a high degree of VNF co-location. We further evaluated a variant of GVB-heuristic-L2, at which the VNF bundling scheme is triggered below a pre-defined resource utilization level. This empowers an InP to properly adjust the trade-off between VNF co-location and resource efficiency in order to increase their revenue by either minimizing resource wastage (*i.e.,* saving capacity to accommodate additional services) or by embedding services at potentially premium prices via a minimal embedding footprint (in order to meet strict latency requirements).

# Chapter 4

# Service Function Chain Embedding for Cross-Service Communication

NFV orchestration in its current form handles individual SFCs separately, leading to limitations in their ability to interact with one another. This lack of interaction becomes a challenge when an SFC requires the consumption of another SFC, or a part of it. In a dynamic service ecosystem that encourages cross-service interactions, it is crucial to consider the orchestration of SFCs while taking their cross-service communication (CSC) requirements into account.

In this chapter, we propose a CSC-aware SFCE heuristic which optimizes the placement of SFCs. It goes beyond considering only the resource and inner-component communication demands and takes into consideration another deployed SFC that needs to be consumed. To address this, we examine different types of CSC and introduce a new data structure which we term *VNF embedding tree*. This data structure helps generate the most suitable sequence for embedding the VNFs of SFCs and enables the heuristic to handle the complexities of SFCE with CSC requirements. Through simulations, we evaluate the efficiency of the proposed heuristic and reveal significant benefits in terms of service co-location, without any noticeable impact on embedding efficiency, when compared to a baseline heuristic that neglects CSC considerations.

## 4.1 Motivation

NFV initially emerged to address network processing needs within the telecommunications sector, specifically by virtualizing middleboxes, such as firewalls, NATs, and proxies [1, 66]. As discussed in Chapter 2, the scope of NFV expanded over time to encompass cellular networks [46, 3] and other applications across various domains, referred to as verticals, including automotive, media, e-health, manufacturing, etc. This evolution has given rise to a dynamic service ecosystem that fosters opportunities for interactions between different services. In this context, one SFC can consume another SFC, thereby enhancing its functionality. For instance, an augmented reality SFC that provides location-based metadata can significantly improve its service delivery by leveraging access to a social network service to offer personalized recommendations. Deploying such SFCs often involves the utilization of dedicated network slices within data centers [16]. This trend is particularly gaining momentum in edge computing to support time-sensitive applications like location-based services [47].

The orchestration of SFCs that need to consume others requires particular care. For example, SFC embedding or scaling should be handled jointly with the consumed SFC. More precisely, the embedding optimization of an SFC does not solely pertain to its own resource/communication demands, but is also pertinent to the placement of the SFC that will be consumed. However, the prevailing way of service orchestration deals with SFCs independent to each other, thereby, performing placement or scaling decisions only with respect to the individual service requirements. This approach can yield sub-optimality in terms of CSC, *i.e.,* the communicating components of the two services may be assigned to datacenter regions (*e.g.,* different racks) with low-bandwidth connection or unnecessary long hop-count, with a potentially adverse impact on cross-service response time.

Along these lines, we emphasize the importance of integrating CSC considerations into SFCE, a problem which we term SFCE-CSC. The goal is to add an additional dimension to the SFCE policy: the co-location of two interacting SFCs. It is imperative for SFCE-CSC to not only factor in the intricate communications and resource needs among its own components, but also to optimize the placement of CSC links. The challenge is further exacerbated when considering the varying nature of CSC, depending on whether a service is fully or partially utilized.

## 4.2 Contributions

Since existing SFCE methods are, in principle, oblivious of CSC, we propose a new heuristic for the SFCE-CSC problem. A key aspect of the proposed heuristic is the *VNF embedding tree*, *i.e.,* a new data structure that facilitates the evaluation of VNF embedding steps, with the aim of generating the most suitable sequence for the embedding of a VNF-graph. Comparing the proposed heuristic with a baseline heuristic designed for generalized SFCE (thereby, not catering to CSC), our proposed method consistently yields a higher degree of co-location of communicating SFCs, albeit not impacting the embedding efficiency of individual SFCs. Concretely:

- We provide a comprehensive mathematical formulation of the SFCE-CSC system model. This formulation establishes a clear framework for understanding and addressing the challenges associated with the embedding of SFCs while considering CSC requirements.
- We present a systematic approach to SFCE-CSC that utilizes innovative techniques. One of the key techniques introduced is the *VNF embedding tree*, which plays a crucial role in optimizing the embedding process. This approach offers a structured and effective method for addressing the complexities of SFCE-CSC.
- We evaluate the proposed method rigorously through extensive simulations. Our assessments demonstrate the effectiveness and efficiency of the approach in terms of service co-location and embedding performance.

## 4.3 Problem Description

Before delving into the SFCE-CSC problem, we lay out our perception of cross-service interactions. To begin with, in the context of CSC, we consider the following SFC classification: (i)

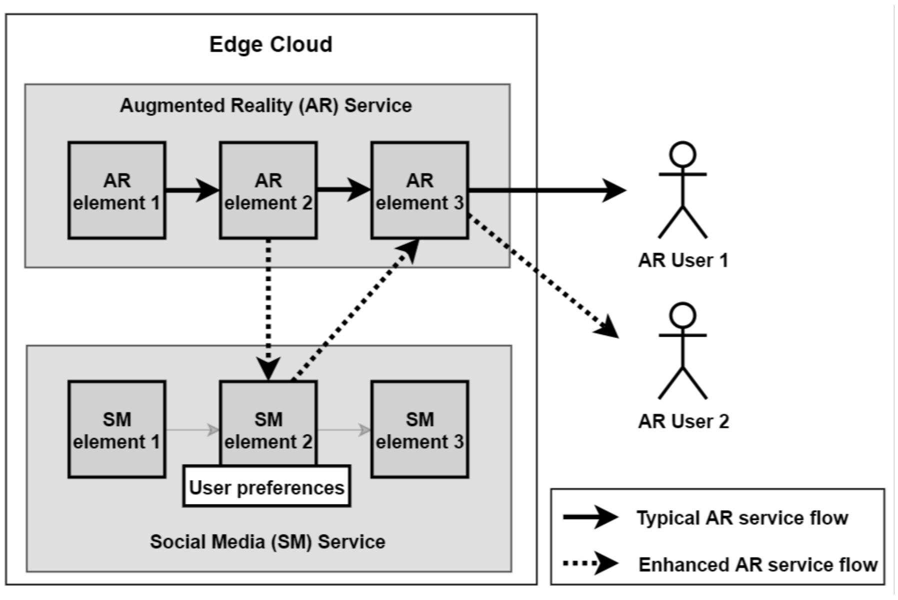


Figure 4.1: A cross-service interaction between an augmented reality SFC, and a social media SFC.

*consuming* SFCs, and (ii) *providing* SFCs. Essentially, CSC provides the means for a consuming SFC to enhance its functionality by gaining access to service elements or functions offered by a providing SFC. Such SFCs may be co-located in the same (edge) cloud infrastructure, opening up an opportunity for peering between the two services [16]. As such, the consuming SFC can be enabled to consume another (*i.e.,* providing) SFC with very low latency, enhancing its functionality and potentially the experience offered to the user. For instance, Fig. 4.1 illustrates an augmented reality (AR) SFC co-located with a social media (SM) SFC. In particular, the AR SFC retrieves user preferences stored in the SM SFC; thus, it can offer an enhanced personalized AR experience to users. In this example, the AR is the consuming SFC, while the SM fulfills the role of the providing SFC. Henceforth, a pair of a consuming and a providing SFC is referred to as a *CSC pair*. This certainly does not preclude a consuming SFC from acting as a providing SFC, and vice-versa, both to the same or to a different CSC pair. In the scope of this chapter, we consider SFCs that fulfil only a single role.

To further exemplify cross-service interactions, we depict Figs. 4.2 - 4.4. Initially, we distinguish between three types of CSC. *CSC type 1* (Fig. 4.2) represents a cross-service interaction, at which a single VNF of the providing SFC is only consumed. In this case, a subset of the traffic traverses only the consuming SFC (*i.e., VNF A* $\rightarrow$ *VNF B*), whereas the rest of the traffic is redirected through the providing SFC (*i.e., VNF A → VNF E → VNF B*). For example, consider a mobile application as the consuming SFC, which offers basic and premium subscriptions. This subscription discrimination exposes basic users to advertisements, while, at the same time, providing an ad-free experience for premium users. In this scenario, *VNF E* of the providing SFC could be a virtualized cache that stores advertising content, which is offered to the consuming SFC. Consequently, traffic associated with basic subscriptions will traverse the path *VNF A → VNF E → VNF B*, instead of the path *VNF A → VNF B*, which will be utilized for premium users.

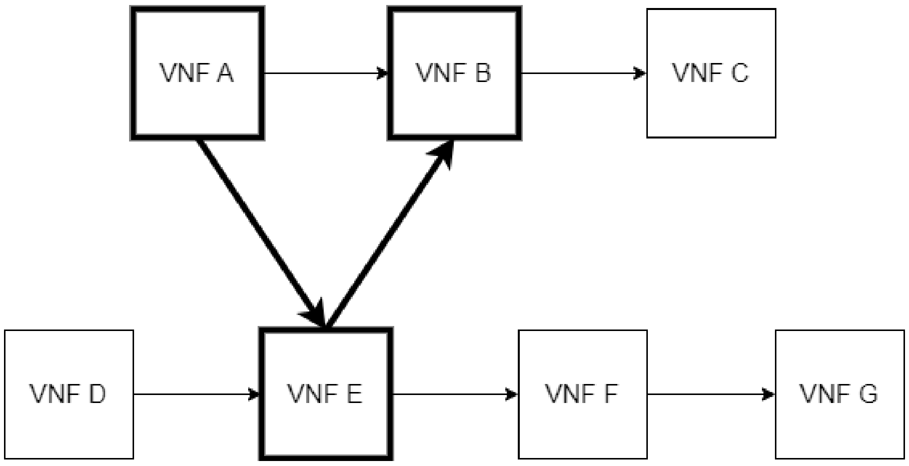


Figure 4.2: The CSC type 1 implies that a single VNF from the providing SFC is consumed.

We further identify an additional CSC type, *i.e., CSC type 2*, at which the consuming SFC accesses a subset of the providing SFC, in the form of a sequence of VNFs, as depicted in Fig. 4.3. This CSC type pertains to SFCs that require the consumption of a wider spectrum of functions offered by a providing SFC. For example, assume that *VNF A* is a firewall, whereas *VNF E* and *VNF F* correspond to a light and heavy intrusion detection system (IDS), respectively. In this case, flows that are suspicious for intrusion can be subjected to an additional two-level inspection by utilizing the respective VNFs of the providing SFC (*VNF A → VNF E → VNF F → VNF B*). Instead, all remaining traffic will traverse only the VNFs of the consuming SFC.

Last, we consider the scenario where the entire providing SFC needs to be consumed, depicted in Fig. 4.4. For instance, assume that the providing SFC is a machine learning application, composed of VNFs that implement data pre-processing, feature engineering, model training and evaluation, which practically constitutes an inseparable pipeline. This case is expressed by

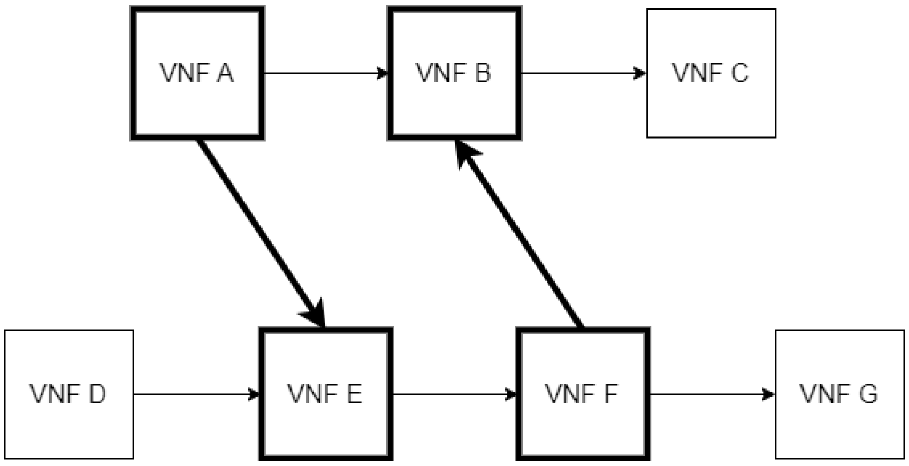


Figure 4.3: In CSC type 2, the consuming SFC accesses two VNFs of the providing SFC.

*CSC type 3*, at which the traffic traverses the entire chain of the providing SFC.

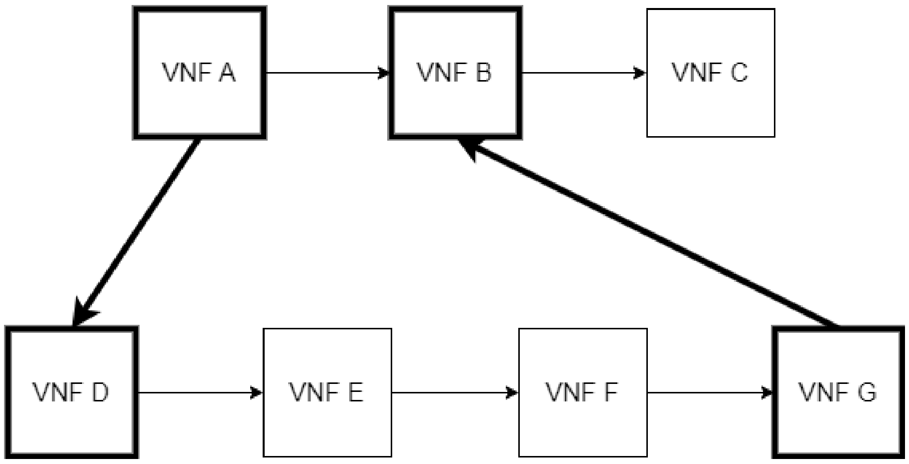


Figure 4.4: CSC type 3 depicts an SFC that consumes the entire providing SFC.

The CSC examples presented above are merely indicative. Various use cases of CSC can be conceived and defined based on existing and future service needs (cf. [16]). Cross-service interactions can significantly enhance service delivery and the user experience, since by exploiting SFC co-location, CSC can be established with minimal delay. However, as we explain in the following, optimizing SFCE, while fulfilling CSC requirements, is not a trivial task.

The need to establish cross-service interactions introduces additional complexity and challenges into the SFCE problem. The main challenge stems from the need to embed *CSC links*, which are marked with bold arrows in Fig. 4.5. These links establish the required connection between the SFCs of a CSC pair. In case both SFCs are instantiated concurrently, we could rely on existing techniques (*e.g.,* [60, 61, 31, 10]) for their embedding, by merely joining the respective VNF-

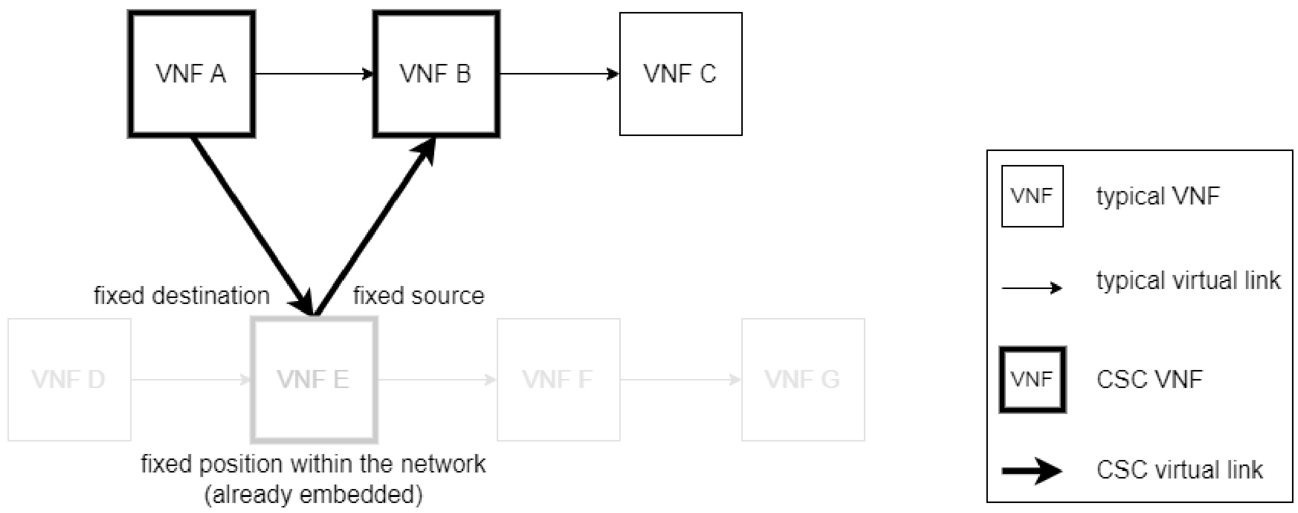


Figure 4.5: In each case, the final SFC to be embedded is expanded by two additional links, whose embedding depends on the positioning of the CSC parts of the pre-embedded providing SFC.

graphs. In reality, however, this is very unlikely; instead, the reasonable assumption is that the providing and consuming SFCs will be deployed asynchronously. More precisely, we consider that the providing SFC is already in place (and consumed by its own dedicated clients), before the deployment of the consuming SFC. As such, the *CSC VNFs* of the providing SFC (*e.g., VNF E* for the CSC type 1) will be already fixed on the underlying network infrastructure. Thereby, the SFCE optimization should handle these as additional constraints in the assignment of *CSC links*, and, in extension, in the mapping of the entire VNF-graph of the consuming SFC.

To explain this further, notice how the VNF-graph of the consuming SFC is expanded in the CSC type 1 scenario. In particular, two CSC links are inserted, which connect *VNF A* with *VNF E*, and *VNF E* with *VNF B*, respectively. Given that *VNF E* is already placed in the network, a single end-point of each CSC link should be attached to a fixed network position, such as a server. This introduces additional embedding limitations, compared to the generalized SFCE problem, which are further exacerbated in CSC types 2 and 3. More specifically, the last two CSC types imply that four VNFs (two VNFs belonging to the providing and the other two VNFs being part of the consuming SFC) are involved in the CSC, whereas three VNFs are engaged in CSC type 1.

## 4.4 Methodology

In this section, we introduce a system model for SFCE-CSC, and further propose a heuristic for SFCE-CSC optimization. Last, we present a variant of this heuristic for the computation of generalized SFCEs (which do not cater to any CSC-related constraints.

### 4.4.1 SFCE-CSC system model

In the problem at hand, we denote the consuming SFC with $G_C = (V_C, E_C)$, and the providing SFC with $G_P = (V_P, E_P)$. According to the discussions in Section 4.3, a CSC path comprises VNFs of both SFCs (*i.e.,* CSC VNFs), as well as links that do not exclusively pertain to any SFC (*i.e.,* CSC links). We model $V_C^* = \{i, j\}$ $(i, j \in V_C, i \neq j)$ as an ordered set that holds the CSC VNFs of $G_C$. Similarly, $V_P^* = \{k, l\}$ $(k, l \in V_P)$ holds the respective VNFs of $G_P$. Note that, in the second case, it could be that $l = k$, corresponding to CSC type 1. We also model the set of CSC links by $E^* = \{(i, k), (l, j)\}$ $(i, j \in V_C^*, k, l \in V_P^*)$. Our base SFC request model (cf. Definition 2.5) augments the modelling of the expanded consuming SFC (ECS), which is as follows:

**Expanded Consuming SFC (ECS) Model.** ECS encompasses the vertices of the initial consuming SFC, plus the CSC VNFs of the providing SFC. Furthermore, ECS extends its edge set by adding the $E^*$ edges, *i.e.,* ECS is modelled as a directed graph $G'_C = (V'_C, E'_C)$, where $V'_C = V_C \cup V_P^*$, and $E'_C = E_C \cup E^*$. Each element of $G'_C$ is associated with certain resource demands, similar to the case of the typical SFC model.

### 4.4.2 SFCE-CSC heuristic

Towards the design of a heuristic algorithm, capable of coping with the complexity of SFCE-CSC, we encounter a crucial issue, which stems from the ambiguity of VNF communication dependencies. That is, the *CSC VNFs* $\in V_C^*$ of an ECS will communicate with both their adjacent VNFs included in $V_C$, as well as with the assigned CSC VNFs of the providing SFC.

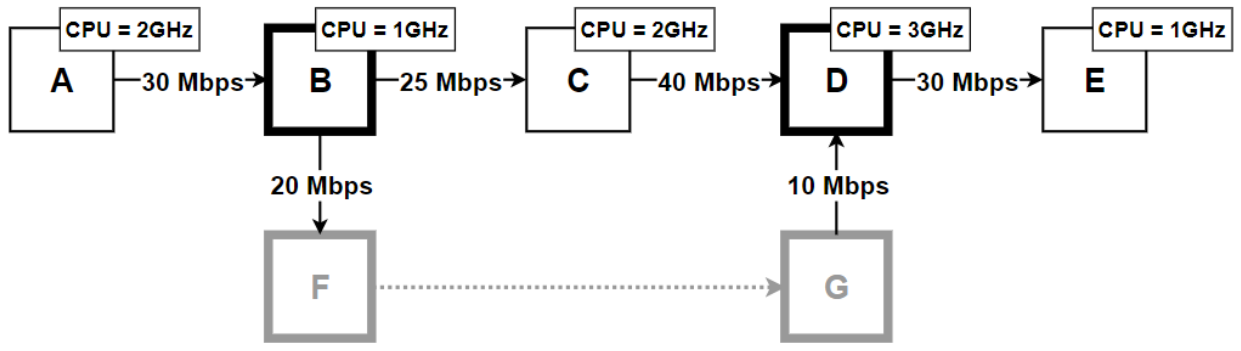


Figure 4.6: VNF dependency ambiguity in $G'_C$.

From an algorithmic design perspective, if we prioritize these dependencies based only on communication requirements (*e.g.,* bandwidth demands), we will neglect a key problem aspect, *i.e.,* that we can capitalize on the mapping flexibility of non-embedded VNFs, while there is not much that can be done for the already assigned VNFs (potential VNF migration among servers could create service disruption and is certainly an operation that can be triggered only at high timescales). For instance, in Fig. 4.6, *VNF B* should be placed in proximity to *VNF A*, *VNF F*, and *VNF C*, which are its adjacent nodes. While its most intense communication occurs with *VNF A*, prioritizing the placement of *VNF B* closer to *VNF F* (which is already embedded in the network), and then trying to map *VNF A* and *VNF C* close to *VNF B*, seems a viable approach to SFCE-CSC.

Following this approach for all VNFs of a consuming service, we face a critical challenge of SFCE-CSC, *i.e.,* how to determine the *VNF embedding sequence*. This term refers to the sequence, according to which, the embedding algorithm will seek to map the VNFs of the consuming service. Apparently, the VNF embedding sequence comprises a fundamental aspect of the overall embedding policy, since constructive heuristics, by definition, have no means of stepping back to modify the (partial) solution (although this feature obviously accelerates the solution computation). This means that, if the global embedding policy does not sufficiently address the problem at hand, the generated embeddings will be inefficient. Back in Fig. 4.6, let us assume that the embedding commences with *VNF A*, which is therefore placed first within the substrate network. This placement, being oblivious to the need of *VNF B* for co-location with *VNF F*, in conjunction with the fact that *VNF F* is already fixed in the network, restricts

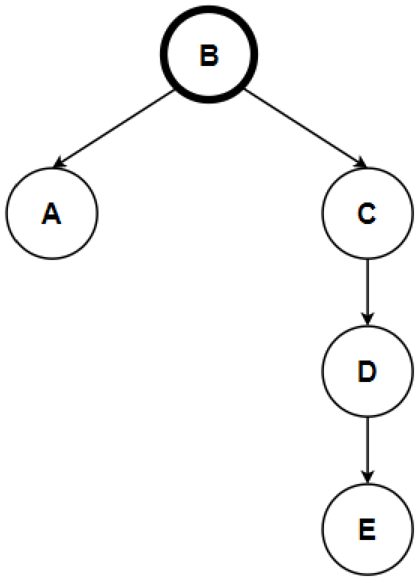


Figure 4.7: VNF embedding tree.

the optimization possibilities.

**VNF Embedding Tree.** Existing heuristics, designed for the generalized SFCE problem, can not efficiently meet the CSC requirements exemplified above. Therefore, we take a different view on the SFCE problem, accounting for the intricacies due to CSC. To this end, we introduce a new structure, namely the *VNF embedding tree*, which is used to generate the VNF embedding sequence. The construction of a VNF embedding tree adheres to the following principles:

1. The root is the CSC VNF of the consuming service with the lowest CPU requirements.
2. The children of a node are its adjacent VNFs $\in V_C$ that are not yet inserted in the tree.
3. Nodes are examined for further branching with a depth-first strategy, prioritizing left edges.
4. Left children have higher communication requirements with their parent, compared to their rightmost siblings.
5. A node is leaf if its parent is its only adjacent node, or if its potential children are already in the tree.

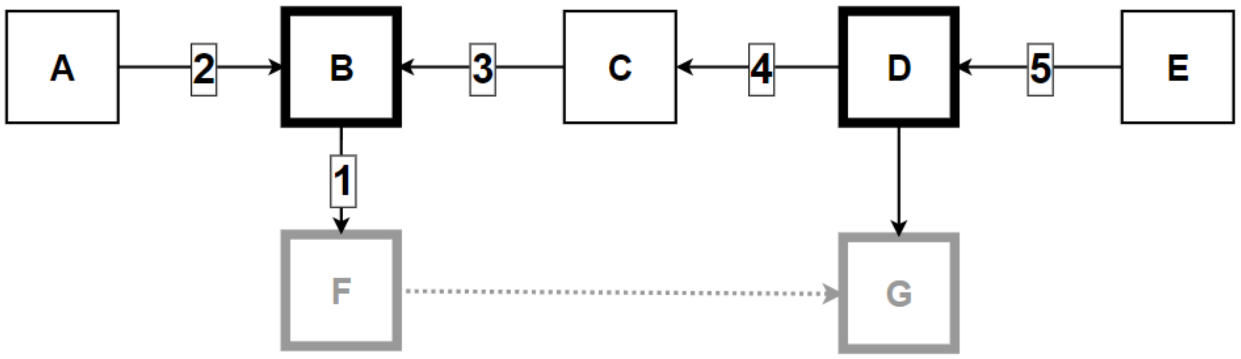


Figure 4.8: VNF embedding sequence derived from the tree of Fig. 4.7.

For instance, given the $G'_C$ shown in Fig. 4.6, *VNF B* will be designated as the root, since $d_{CPU}(B) = 1GHz < 3GHz = d_{CPU}(D)$. This design decision (*i.e.,* principle 1) stems from our intuition that, keeping a node $X$ with low CPU requirements at higher levels of the tree, increases the probability of co-locating the nodes of entire sub-trees rooted at $X$. The next step is to obtain *VNF B*'s adjacent VNFs (that belong to the consuming service). These are *VNF A* and *VNF C* and, thereby, these are added in the tree as *VNF B*'s children. Given that $d(A, B) = 30Mbps > 25Mbps = d(B, C)$, *VNF A* will be *VNF B*'s left child, whereas *VNF C* will be its right child. No further branching occurs below *VNF A*, since its only adjacent node is its parent, *i.e., VNF B*. On the other hand, there is a link between *VNF C* and *VNF D*; therefore, the latter is a child of the former. Similarly, *VNF E* is placed below *VNF D*, and this completes the tree depicted in Fig. 4.7. Note that the proposed tree structure can be applied on non-linear consuming service graphs, as well.

**Inferring the VNF embedding sequence.** Previously, we mentioned that the embedding tree is employed to derive the VNF embedding sequence that captures the fundamental properties of SFCE-CSC. Essentially, this is realized through the following logic: *the root of the tree will always be placed towards the already embedded node* $\in V_P^*$ *with which it communicates, while the rest of the nodes will be placed towards their parent.*

The VNF embedding sequence is dictated by the visiting order of the tree's nodes, when these are traversed with a depth-first strategy, prioritizing left sub-trees. This, in conjunction with the core logic described above, will result in the embedding sequence illustrated in Fig. 4.8 for the tree of Fig. 4.7. In particular, the heuristic will initially seek to place *VNF B* as proximate to *VNF F* as possible. Subsequently, it will try to co-locate *VNF A* with *VNF B*, and then *VNF C* with *VNF B*. Likewise, *VNF D* will be sought to be mapped close to *VNF C*. Finally, *VNF E*'s mapping will depend on the placement of *VNF D*.

**VNF co-location policy.** The proposed embedding heuristic strives to co-locate *VNF X* towards an already placed *VNF Y*, meaning that it will first attempt to place $X$ in the same server that $Y$ resides in. If this is not possible, the heuristic will attempt to embed $X$ in a server belonging to the same rack; otherwise, it will place $X$ in a server within the rack with the least outbound inter-rack traffic. The candidate servers of a rack are selected with a *Worst Fit* logic, if $X$ is not a leaf node, since additional VNFs will pursue their co-location with $X$ (*i.e.,* $X$'s children). If $X$ is leaf, a *Best Fit* approach is followed, since no additional VNFs will seek to communicate with $X$, thus, the algorithm should allocate CPU as efficiently as possible. This procedure is described in detail in Algorithm 2.

### 4.4.3 Baseline heuristic

The *baseline* heuristic maps an SFC into a datacenter as follows. Initially, it sorts the racks of the datacenter in descending order, according to their average available ToR-to-core switch link bandwidth. Commencing with the first ordered rack, it ranks its servers in descending order, according to their available CPU capacity, and strives to place VNFs sequentially, starting from the first VNF of the chain. The VNF co-location policy is exercised in the same way with the SFCE-CSC heuristic.

The baseline heuristic takes no particular actions for the optimization of SFCE-CSC; rather it focuses on minimizing the embedding footprint of the consuming SFC, without any consideration for the CSC links. The main purpose of the baseline algorithm is to let us quantify the potential gains of our proposed SFCE-CSC, by comparing the efficiency of the two heuristic variants.

**Algorithm 2** SFCE-CSC VNF placement

```
1: Input: $sequence$, $datacenter$
2: Output: $placed$
3: for $node \in sequence.keys()$ do
4:    $placed = False$
5:    $target\_server$ = the server where the $sequence[node]$
      node resides in
6:    # try to place in the same server
7:    if $node$ fits in $target\_server$ then
8:       $target\_server.place(node)$
9:       $placed = True$
10:      continue
11:   # try to place in the same rack
12:   if not $placed$ then
13:      $target\_servers$ = the rest of the servers in the rack
         of the $target\_server$
14:      $isLeaf = True$ if the node is leaf, $False$ otherwise
15:      $target\_servers.sort(reverse = isLeaf)$
16:      for $s \in target\_servers$ do
17:         if $node$ fits in $s$ then
18:            $s.place(node)$
19:            $placed = True$
20:            break
21:   # try to place in different racks
22:   if not $placed$ then
23:      for $rack \in racks.sort()$ do
24:         $target\_servers$ = the servers of the current rack
25:         $target\_servers.sort(reverse = isLeaf)$
26:         for $s \in target\_servers$ do
27:            if $node$ fits in $s$ then
28:               $s.place(node)$
29:               $placed = True$
30:               break
31:         if $placed$ then
32:            break
33:   if not $placed$ then
34:      return $placed$
35: return $placed$
```

Table 4.1: Datacenter Parameters

| | |
|---|---|
| Number of core switches | 5 |
| Number of racks | 10 |
| Number of servers / rack | 20 |
| Server CPU capacity | 7.2 GHz |
| Intra-rack link capacity | 1 Gbps |
| Inter-rack link capacity | 10 Gbps |

Table 4.2: SFC Request Parameters

| | |
|---|---|
| Number of VNFs / SFCE | U[3,8] |
| VNF CPU requirements | U(.1,.2,.3,.4,.5) · 7.2 GHz |
| Link bandwidth requirements | U[10,100] Mpbs |

Table 4.3: Evaluation Environment Parameters

| | |
|---|---|
| SFCE requests / time interval | Poisson(20) |
| SFC type | expiring |
| SFC lifespan | U[3,10] time intervals |
| Number of time intervals | 600 |

## 4.5 Evaluation Results

In this section, we perform a comparison between the proposed SFCE-CSC heuristic and the baseline variant, using simulations. Initially, we present our evaluation environment and metrics and, subsequently, we proceed with the discussion of our evaluation results.

### 4.5.1 Evaluation Setup

We have developed a simulation environment for SFCE evaluations in Python [67]. All evaluations are carried out on a two-layer fat-tree datacenter network topology. The datacenter consists of 200 servers, which are evenly arranged into 10 racks. The corresponding 10 ToR switches are interconnected through 5 core switches. The intra-rack and inter-rack links have capacity of 1 Gbps and 10 Gbps, respectively. The CPU capacity of each server is set to 7.2 GHz (cf. Table 4.1).

In order to evaluate the SFCE-CSC efficiency, we pre-embed ten providing SFCs into the datacenter, with each one placed within a single rack. In this way, we do not preclude the existence of efficient solutions (which would be the case, if the VNFs of a providing SFC span multiple racks). In this respect, we generate consuming SFCs that comprise 3 to 8, sequentially connected, VNFs. Each VNF requires 10, 20, 30, 40 or 50% of a server's CPU in order to process its inbound traffic. We assume that bandwidth requirements vary for each virtual link in the range of 10 Mbps to 100 Mbps[1]. Furthermore, each consuming SFC requires communication with a single providing SFC, which can be of either type 1, 2 or 3, as described in Section 4.5. The aforementioned parameters, summarized in Table 4.2, are obtained from uniformly random distributions.

Towards a more realistic simulation environment, we account for a set of $n$ discrete time steps $T = \{1, \dots, n\}$. At each time step, a number $N$ of consuming SFCs request placement sequentially, where $N \sim Poisson(20)$. Each SFC comes with a lifespan $k$. Therefore, if an SFC is embedded at time $t \in T$, it will expire at time $t + k \in T$ (*i.e.,* it will be decommissioned from the datacenter). During our simulations, $k$ lies within $[3, 10]$ (randomly), and $n = 600$ (cf. Table 4.3).

We compare the two heuristic variants based on the following criteria: (i) co-location efficiency of adjacent VNF pairs (*i.e.,* how proximate two communicating VNFs are placed), (ii) embedding efficiency of CSC and non-CSC links, (iii) amount of inter- and intra-rack generated traffic, (iv) request acceptance rate (ratio of successfully embedded SFCs over the total number of embedding requests), and (v) CPU utilization, which corresponds to the amount of the total CPU allocated to all embedded SFCs.

### 4.5.2 Comparison of SFCE-CSC and baseline

Throughout this section, the terms SFCE-CSC and NSE-CSC (where the latter is typically encountered in plot legends) are used interchangeably.

We commence by discussing the VNF co-location efficiency. To this end, we examine the proximity of each pair of adjacent VNFs (*i.e.,* VNFs connected with a virtual edge), after

[1]Such variations have been thoroughly examined in, *e.g.,* [68], but are out of the scope of this work.

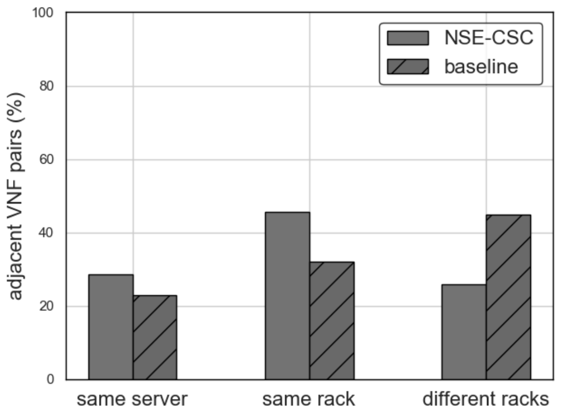


Figure 4.9: Distribution of adjacent VNF pairs into same server, same rack, and different racks.

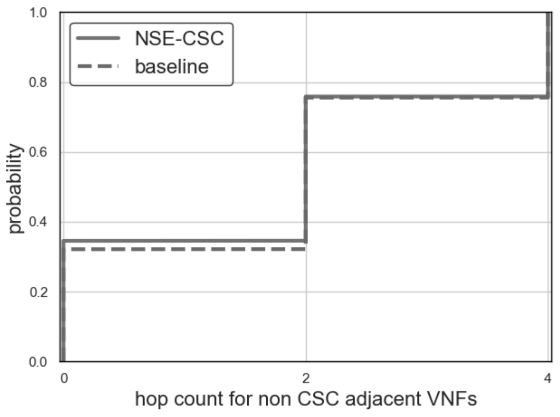


Figure 4.10: CDF of the hop count for non-CSC adjacent VNFs.

their placement within the substrate network. According to Fig. 4.9, the SFCE-CSC heuristic maps ≈ 30% of such VNF pairs into the same server, whereas the respective percentage for the baseline method is slightly above 20%. Regarding the VNF pairs mapped into the same rack, the SFCE-CSC heuristic outperforms the baseline variant, since the corresponding percentages exhibit a difference greater than 10%. We further examine the mappings of communicating VNFs in different racks. It becomes apparent that the baseline method unfavourably maps nearly half of such VNFs into different racks, while only a quarter falls into the same category for the proposed heuristic. Apparently, this phenomenon is attributed to the superior capability of the SFCE-CSC algorithm to handle the optimized placement of CSC links, as opposed to the baseline method that merely optimizes the placement of the consuming service graph.

In order to provide more insights from the previous results and also identify whether the proposed algorithm skews in favour of the embedding optimization of the CSC links to the detriment of the embedding optimization of non-CSC links, we present Figs. 4.10 and 4.11. In particular, Fig. 4.10 illustrates the CDF of the hop count of non-CSC links, *i.e.,* the edges solely pertaining to the consuming service graph. We note that the two-layer hierarchical topology used throughout our simulations implies three possible hop count values for an edge connecting two VNFs, *i.e.,* 0 (when the VNFs are embedded onto a single server), 2 (when the VNFs are embedded into different servers of the same rack), and 4 (when the VNFs are embedded

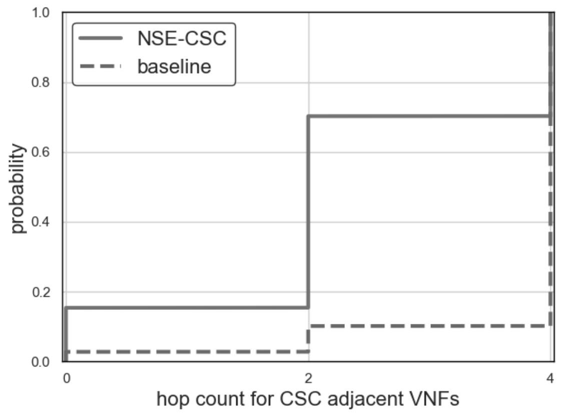


Figure 4.11: CDF of the hop count for CSC adjacent VNFs.

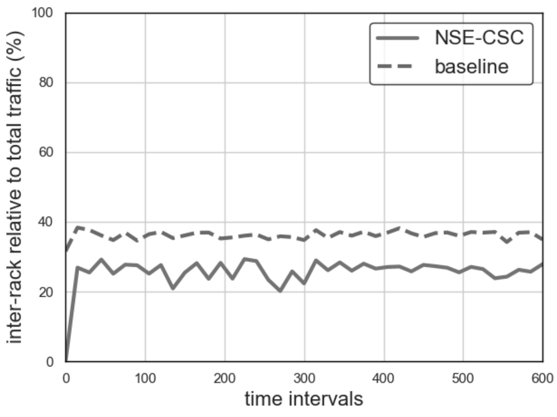


Figure 4.12: Ratio of inter-rack traffic to total traffic (total = inter-rack + intra-rack traffic).

into servers that belong to different racks). Fig. 4.10 indicates a similar behaviour of the two methods, with respect to the placement efficiency of non-CSC links. This is very important, since the embedding efficiency of the consuming service part of the ECS graph is similar for both methods, as most of the time (*i.e.,* ≈ 80%) the virtual links are confined within a single rack.

However, according to Fig. 4.11, the heuristic variants perform differently on the mapping of CSC links. More specifically, with the SFCE-CSC method, ≈ 75% of these links are mapped on paths comprising at most two physical links, meaning that the respective VNF pairs are placed either on the same server or on different servers within the same rack. The corresponding percentage resulting from the baseline algorithm is roughly 10%, *i.e.,* 9 out of 10 pairs that comprise both consuming and providing VNFs are placed across different racks. This alone confirms the insights gained from Fig. 4.9 about the baseline method, regarding the high percentage of VNF pair distribution into different racks. We should note that the results plotted in Figs. 4.9, 4.10, and 4.11 are obtained from a single snapshot of the infrastructure state, at the same time interval for both methods. However, our experimentation corroborates that similar patterns hold in general.

The previous results indicate that SFCE-CSC exhibits a higher degree of intra-rack consolidation, compared to the baseline. This is also corroborated by Fig. 4.12, where the ratio of

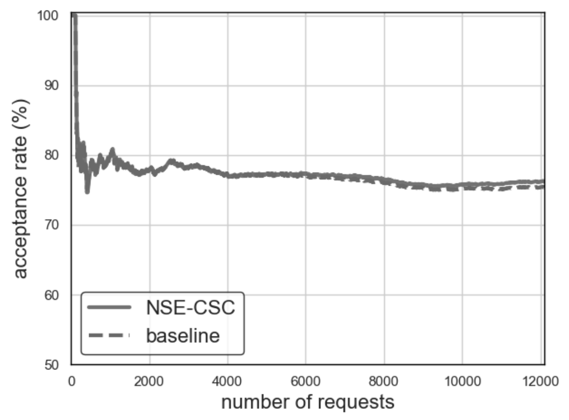


Figure 4.13: Acceptance rate of the SFCE-CSC and the baseline heuristic.

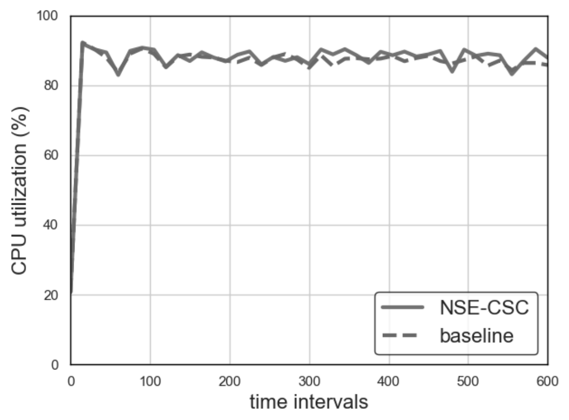


Figure 4.14: CPU allocation for the SFCE-CSC and the baseline heuristic.

inter-rack to total traffic (*i.e.,* total traffic = inter-rack traffic + intra-rack traffic) across all time intervals is depicted, for both methods. The main insight here is the considerable gap (*i.e.,* $\approx$ 10%) in terms of generated inter-rack traffic (relative to the total traffic) between the SFCE-CSC and the baseline. Such inter-rack bandwidth conservation by SFCE-CSC allows for higher levels of datacenter network over-subscription and, in turn, cheaper switching hardware at the core level of the datacenter topology. Besides the cost reduction, confining traffic within a rack can lead to lower service response time and more predictable performance for an SFC (*e.g.,* lower probability for Service Level Agreement violations).

Last, we investigate the efficiency of the two heuristics in terms of acceptance rate and CPU allocation. Figs. 4.13 and 4.14 indicate that both methods exhibit similar levels of efficiency with respect to these criteria, with only marginal differences that slightly favour the SFCE-CSC. The small gain of the SFCE-CSC potentially stems from the way it alters the ranking strategy of servers (*Worst / Best Fit*), since the baseline method does not incorporate any such feature.

According to our evaluation results and the aforementioned discussions, the proposed SFCE-CSC heuristic manages to optimize the placement of ECSes, without any penalty on the embedding efficiency of the consuming service part of the graph, or the resource allocation efficiency of the substrate network. Considering also the achievable optimization of the embedding of CSC links, we deem SFCE-CSC as a viable and efficient solution to the SFCE-CSC problem.

## 4.6 Related Work

Authors in [69] deal with SFCE for diversified 5G slices, considering the feature of sharing VNFs among different slices. Besides an integer linear programming (ILP) formulation for the problem, an efficient heuristic algorithm is also provided to cope with its computational complexity. *Papagianni et al.* [3] also promote the sharing of VNFs across multiple network services, but this time, in the context of LTE slicing, in order to reduce provisioning and management costs. To this end, the authors propose a mixed integer linear programming (MILP) formulation for SFCE with shared VNFs.

Another solution to SFCE with VNF sharing is FlexShare [70], which comprises a four-step mechanism for the optimized deployment of an SFC. Initially, a bipartite graph is constructed, including the VNFs to be deployed at one side, and the potential virtual machines (VMs) that can host them on the other, with links between them indicating feasible assignments. The Hungarian algorithm is employed to optimize the latter, during the second step. The subsequent procedure deals with prioritization and computational capacity allocation. If these cannot be satisfied, FlexShare triggers the fourth step, which prunes the original bipartite graph, and repeats steps two and three until a feasible solution has been obtained.

Although the aforementioned studies are relevant to our work, their view on VNF sharing is considerably different to the notion of CSC, which, to the best of our knowledge, has not been studied so far. In particular, the above consider the shared VNFs as an indispensable part of the SFC graph that needs to be embedded, and, if needed (*i.e.,* if resources of instantiated VNFs are not sufficient for sharing), new instances can be spawned within the network infrastructure. In contrast, CSC implies that the shared VNF(s) are exclusive property of a unique SFC (*i.e.,* the providing SFC), hence a consuming SFC has no means of replicating them on demand within the network. This fundamental observation renders existing SFCE solutions insufficient for SFCE-CSC.

## 4.7 Conclusions

We tackled the challenging problem of SFCE-CSC, *i.e.,* the optimization of SFCE subject to CSC requirements. To this end, we designed a new heuristic that introduces a new policy dimension to generalized SFCEs, *i.e.,* the co-location of the pair of providing and consuming SFCs. Since the CSC links create placement dependencies, particular attention is required in the design of any SFCE-CSC heuristic, with respect to the sequence of VNF embedding, such that the overall optimization is tailored to both CSC and typical SFCE requirements. Driven by this key insight, we introduced the VNF embedding tree in order to derive the most suitable VNF embedding sequence, and, thereby, dictate the embedding steps that the heuristic should follow for optimized SFCE-CSC.

Our evaluation results uncover significant gains for the proposed heuristic in terms of service co-location. The proposed heuristic, most of the times, achieves the co-location of the CSC pair within a single rack (and sometimes within the same server), ensuring lower cross-service response times, which, in turn, can minimize SLA violations. Most importantly, these gains do not come at the expense of any embedding inefficiencies. More precisely, SFCE-CSC is on par with the baseline in terms of acceptance rate and overall CPU allocation.

The advancements presented in the SFCE-CSC solution bear significant contributions to the broader realm of network optimization. By effectively addressing the SFC co-location challenge, we have laid the groundwork for more efficient, reliable, and responsive network service frameworks. An immediate application of SFCE-CSC can be seen in its potential to augment the cross-slice communication framework as proposed in [16], specifically in the placement of intercommunicating network slices. This can lead to not only enhanced user experience due to faster response times, but also to more sustainable and cost-effective network operations.

# Chapter 5

# Service Function Chain Embedding with Multi-Agent Reinforcement Learning

As already discussed in Chapter 2, there is an apparent tendency in the telco industry towards network automation. This affects SFC orchestration to a great extent, as the relevant literature exhibits a gradual shift towards data-driven SFCE frameworks, primarily based on deep reinforcement learning (RL).

In this chapter, we initially elaborate on the reasons supporting the advent of RL within the SFCE problem space, and we then identify crucial limitations of existing RL-based SFCE approaches. In particular, we argue that most of them stem from the centralized implementation of RL schemes. Therefore, we devise a cooperative deep multi-agent reinforcement learning (DMARL) scheme for decentralized SFCE, which fosters the efficient communication of neighboring agents. Our simulation results (i) demonstrate that DMARL outperforms a state-of-the-art centralized *double deep Q-learning* algorithm, (ii) unfold the fundamental behaviors learned by the team of agents, (iii) highlight the importance of information exchange between agents, and (iv) showcase the implications stemming from various network topologies on the DMARL efficiency.

## 5.1 Motivation

Recent *zero touch network and service management* [71, 72] initiatives in the telecommunications industry aim to streamline complex network operations and management tasks in an automated fashion. Traditionally, decision-making systems in this domain rely on heuristics and linear programming techniques, such as MILPs. However, these approaches have limitations in handling the complexity and dynamic nature of modern networks. This is where data-driven frameworks, such as RL, offer significant advantages. Specifically:

**Complexity handling.** Telecommunication networks have become highly complex, with numerous interconnected elements, diverse services, and dynamic traffic patterns. Heuristic and MILP-based approaches often struggle to effectively handle this complexity. This can stem from lack of modeling accuracy, scalability limitations, etc. In contrast, RL can learn from vast amounts of network data and adapt to complex scenarios. More importantly, it can learn the underlying connection of raw system metrics (which describe arbitrary network states) to generic optimization objectives (which drive actions towards desirable network states), making it better suited to handle intricate network management tasks.

**Uncertainty handling.** Telecommunication networks often face uncertain and unpredictable conditions, such as fluctuations in user demand or equipment failures. Heuristic and MILP approaches may struggle to handle such uncertainties effectively. RL, by learning from historical data and exploring different actions, is capable of devising robust decision making policies by accounting for such uncertain conditions, thus optimizing network performance under varying circumstances.

**Self-learning and improvement.** Heuristics and MILP models require manual design and frequent updates to account for network changes. Conversely, RL enables intent-based networking, where the user merely specifies some quantitative objective (*e.g.,* in the form of a service level objective), and the RL algorithm shall learn from network data to adapt its policy on par with the changes within the network. This reduces the need for human intervention and manual updates.

According to the above, it is only natural that RL algorithms are gaining traction in the landscape of SFCE methods. In particular, RL has been mostly studied in the context of SFCE over multi-PoP networks (these are described in Section 2.4.2). In the multi-PoP substrate setting, the decision-making process typically spans the entire hierarchy of an NFV MANO system, since the servers of each PoP are managed by at least one VIM, and SFCs spanning multiple VIMs are managed by the NFVO (cf. Fig. 2.1). Hence, it is crucial to acknowledge the distinctiveness of the NFVO layer to the VIM layer, along with the implications it has with respect to SFCE. Studying the relevant literature through this lens, we have identified certain limitations, the most crucial of which are listed and discussed below:

**Association of intents.** A critical limitation of centralized SFCE with RL is the association of the resource allocation intents of the NFVO with the respective intents of the underlying VIMs. That is, as it is common in hierarchical resource management systems with global and local controllers (*e.g.,* NFVOs and VIMs, VIMs and hypervisors, k8s master and worker nodes), the global controller queries the local controllers about their available resources, and uses this information to compute a resource allocation decision (which is eventually realized by the latter). Effectively, this limits the inherent capacity of the local controllers to express their own resource allocation intents. In fact, global and local intents are not necessarily conflicting; on the contrary, fostering the acknowledgement of local intents can improve the resource allocation efficiency of the overall system, since each local controller has a more precise view of its actual state.

**Decision-making with incomplete information.** The common assumption that a single learning agent, positioned at the NFVO layer (which is the standard practice in centralized SFCE with RL), has a precise view over the entire substrate, is somehow unrealistic. In fact, the higher we move along the NFV MANO hierarchy, the more coarse-grained the information we have at our disposal (because of data aggregation), which implies higher uncertainty. Conversely, if a centralized RL approach admits partial observability, it follows that the decision-making process relies on incomplete information.

**Dependence of action space on topology nodes.** In most works that propose centralized

RL schemes for the SFCE problem, the action space of the learning agent coincides with the set of available PoPs. This is somewhat natural, since the centralized agent has to infer which is the best PoP to host a particular VNF. However, the dependence of the agent's architecture on the physical nodes of the substrate makes the RL scheme particularly hard to scale. That is, the larger the action set, the longer the training time. Our argument is further strengthened by studies which employ techniques for shrinking the action space, *e.g.,* clustering a set of PoPs into a single group, or placing the entire SFC within a single PoP (*e.g.,* [73, 74]).

The above highlight the necessity for tailored RL-driven SFCE methods that are better grounded to real-world aspects.

## 5.2 Contributions

Admittedly, targeting a single solution that addresses all of the aforementioned limitations would be extremely optimistic. However, one can easily observe that most shortcomings stem from the centralized implementation of SFCE. Indeed, a decentralized approach that assigns learning agents locally to each VIM, in conjunction with a module at the NFVO layer that coordinates the resulting local decisions, would alleviate many of these limitations. In particular, (i) the global and local intents would be naturally decoupled, (ii) each agent would act based on detailed local observations; thus, from the perspective of the overall system, the true state would be fully observable, and (iii) it would be easier to decouple action spaces from the physical topology. Along these lines, this chapter focuses on the investigation, the development, and the evaluation of a decentralized SFCE scheme based on cooperative deep multi-agent RL (DMARL). Our main contributions are the following:

- We devise a DMARL framework for decentralized SFCE. Our DMARL algorithm consists of independent *double deep Q-learning* (DDQL) agents, and fosters the exchange of concise, yet critical, information among them.
- We develop a DDQL algorithm for centralized SFCE, and compare its performance with

DMARL.

- We quantify fundamental rules identified by the team of agents over the course of training.
- We examine the impact of (imperfect) cross-agent communication across various substrate network topologies.

## 5.3 Reinforcement Learning Primer

This section commences with an introduction to the core concepts of single-agent RL, and then proceeds to a discussion of the cooperative case of multi-agent RL. Finally, we present the DDQL algorithm, which serves as the basis of the SFCE frameworks in this chapter.

### 5.3.1 Single-agent Reinforcement Learning

The typical RL setting considers a learning-agent, an environment, a control task subject to optimization, and a discrete time horizon $H = \{1, 2, ...\}$, which can as well be infinite. At time $t \in H$, the agent observes the current state of the environment $s_t \in S$, with $S$ indicating the entire set of possible states. Equipped with a set of actions $X$, the agent shall interact with the environment by choosing $x_t \in X$. Then, the agent receives feedback regarding the quality of its action via a reward signal $r_t$, and the environment transitions to the subsequent state $s_{t+1} \in S$.

Conventionally, an RL task is modelled as a Markov decision process, defined by the tuple $< S, X, S_1, T, R >$. Here, $S$ and $X$ are as before. $S_1 \in P(S)$ denotes the starting state distribution, and $T : S \times X \to P(S)$ is the state transition function, where $P(\cdot)$ denotes the set of probability distributions over a set. Finally, $R : S \times X \times S \to \mathbb{R}$ expresses the reward function, which is generally denoted as $R_t$ instead of $R(s_t, x_t, s_{t+1})$ for simplicity. Naturally, the aim of the agent is to maximize the accumulated discounted reward over $H$, *i.e.*, $\sum_{t \in H} \gamma^t R_t$, where $\gamma \in [0, 1)$ is a discount factor employed to prioritize immediate reward signals against those projected farther into the future. To achieve that, the agent needs to perform a sequence of actions based on a learned *policy* $\pi$, which is practically a function that maps states to probability distributions over actions, *i.e.,* $\pi : S \to P(X)$, and the optimal policy is denoted as $\pi^*$.

An important concept here is the *action-value function*, which quantifies the agent's incentive of performing a specific action on a particular state:

$$Q(s_t, x_t) := \mathbb{E}_t \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k} | s_t, x_t \right], (s_t, x_t) \in S \times X \tag{5.1}$$

Apparently, if the agent explores its action selection on the environment long enough such that $Q$-values represent ground truth reward values, then finding the optimal policy becomes trivial. That is, $\pi^*(s_t) = argmax_{x \in X} Q(s_t, x)$. However, $Q$-values are treated as estimates, since the assumption of perfect exploration is hardly ever realistic. To this end, the agent needs to balance *exploration* and *exploitation*, where the former shall strengthen its confidence on $Q$ estimations, and the latter will enable it to benefit from accumulated knowledge. The most common approach to achieve a favourable exploration-exploitation trade-off is via an $\epsilon - greedy$ action selection strategy. According to $\epsilon - greedy$, the agent performs a random action with probability $\epsilon$, while with probability $1 - \epsilon$ the action with the highest $Q$-value is selected. Given an initial $\epsilon$, *i.e.,* $\epsilon_0$, and a decay factor $\epsilon_{decay} \in (0, 1)$, the agent can gradually shift from exploratory to exploitative:

$$\epsilon_t = \epsilon_0 \cdot (\epsilon_{decay})^t \tag{5.2}$$

### 5.3.2 Cooperative multi-agent Reinforcement Learning

MARL builds upon the fundamental blocks of single-agent RL. In particular, MARL considers a set of $n$ agents $A = \{1, ..., n\}$ interacting with the environment. At each time step $t \in H$, each agent $\alpha \in A$ observes $o_t^\alpha$, and draws an action $x_t^\alpha$ from its own designated action set $X^\alpha$. The joint action $\boldsymbol{x_t} = (x_t^1, ..., x_t^n)$ is then applied on the environment, which transitions to a new state $s_{t+1}$. In extension, each agent $\alpha$ observes $o_{t+1}^\alpha$. Similar to single-agent RL, the transition $(s_t, \boldsymbol{x_t}, s_{t+1})$ is evaluated by means of a reward function $R$, and a scalar $r_t^\alpha$ is sent to each agent. In a fully cooperative MARL setting, all agents share the same reward, *i.e.,* $r_t^\alpha = r_t,\ \forall\ \alpha \in A$.

In principle, the dynamics of the above scheme can be captured by a *cooperative Markov game*, typically defined by the tuple $< S, \boldsymbol{X}, S_1, T, R, Z, O, n >$. Here, $S$, $S_1$, $T$ and $R$ are as before. $\boldsymbol{X} = X^1 \times ... \times X^n$ denotes the joint action space, $Z$ expresses the space of observations, and $O : S \times A \to Z$ is the observation function that dictates the partial observability of each agent (*i.e.,* $O$ can be seen as the function that maps a state $s_t$ and an agent $\alpha$ to an observation $o_t^\alpha \in Z$). In this setting, the goal of each agent $\alpha$ is to discover a policy $\pi^\alpha : Z \to P(X^\alpha)$, such that the joint policy[1] $\boldsymbol{\pi} = (\pi^1, ..., \pi^n) : S \to P(\boldsymbol{X})$ maximizes the (discounted) accumulated reward.

Undoubtedly, the simplest way of establishing a cooperative MARL framework is to treat each learning agent as an independent learning module. In this setting, if each agent is realized as a $Q$-learning agent, the respective MARL system is known as *independent Q-learning* (IQL) [75]. A major limitation of IQL is its lack of convergence guarantees, primarily stemming from the fact that the environment is non-stationary from the perspective of each agent. Effectively, this means that for an agent $\alpha$, both its reward $r_t$ and its next observation $o_{t+1}^\alpha$ are not solely conditioned on its current observation $o_t^\alpha$ and action $x_t^\alpha$. In other words, the transitions of the environment, and, in extension, the common rewards, are affected by the actions of other agents as well. Even though obtaining an optimized joint policy $\boldsymbol{\pi}$ while agents treat other agents as part of the environment is seemingly difficult, IQL exhibits good empirical performance [76].

Cooperative MARL is an active research field, given that numerous control tasks can be seen through the lens of multi-agent systems. In the context of collaborative DQL agents, recent works (*e.g.,* [77, 78]) have established frameworks that promote joint training of multiple agents (*i.e., centralized training - decentralized execution*), under the assumption that the $Q^{total}$-

[1] Here, we implicitly assume that the union of partial observations of all agents can compose the state space, *i.e.,* $\cup_{\alpha \in A} o_t^\alpha = s_t, \forall t$.

function of the multi-agent system can be decomposed into individual $Q$-functions such that, if each agent $\alpha$ maximizes its own $Q^{\alpha}$, then $Q^{total}$ is also maximized.

Irrespective of the promising advances in the field, our work builds upon typical IQL for the following reasons. First, IQL is simple enough to facilitate the interpretation of the system's learning behavior, as it avoids complex NN training schemes. Second, as it is apparent in Section 5.4.2, we envisage an action *coordination* module at the NFVO layer to handle the constraints of SFCE, which can significantly benefit the overall IQL framework.

### 5.3.3 Double Deep Q-Learning

A well-known method that leverages the notions above is *Q-learning*, which is a *model-free*[2], *off-policy*[3] RL algorithm. $Q$-learning attempts to find optimal policies via a direct estimation of $Q$-values, which are maintained in a tabular format. Each time a state-action pair $(s_t, x_t)$ is visited, the respective $Q$-value is updated. An apparent restriction here is the lack of generalization capacity. That is, in problems with large state-action spaces, the risk of scarce $Q$-updates is high.

To overcome this limitation, *Mnih et al.* [79] propose a $Q$-function approximation scheme based on deep neural networks (NNs), commonly termed as deep $Q$-learning (DQL). As per DQL, the learning capacity of the agent lies in an NN which approximates the $Q$-function by parameterizing it, *i.e.,* $Q(s, x) \sim Q(s, x; \theta)$ (where $\theta$ represents a vector of weights and biases of the NN). Ultimately, given a state as input, the NN computes a $Q$-value for every possible action. Then, it is up to an *action selection strategy* (*e.g.,* $\epsilon - greedy$) to choose a particular action.

Two mechanisms that contribute to the success of DQL are the *replay memory* and the coordination of an *online* NN and a *target* NN [80]. The replay memory is used to store *state-action-reward-next state* transitions; with a proper sampling over it, we can subsequently train the online NN with uncorrelated data. The interplay of the two NNs works as follows: initially, the target NN is an exact replica of the online NN, meaning that they share the same architecture,

[2]It does not intend to discover the transition function.

[3]While following a specific policy, it assesses the quality of a different one.

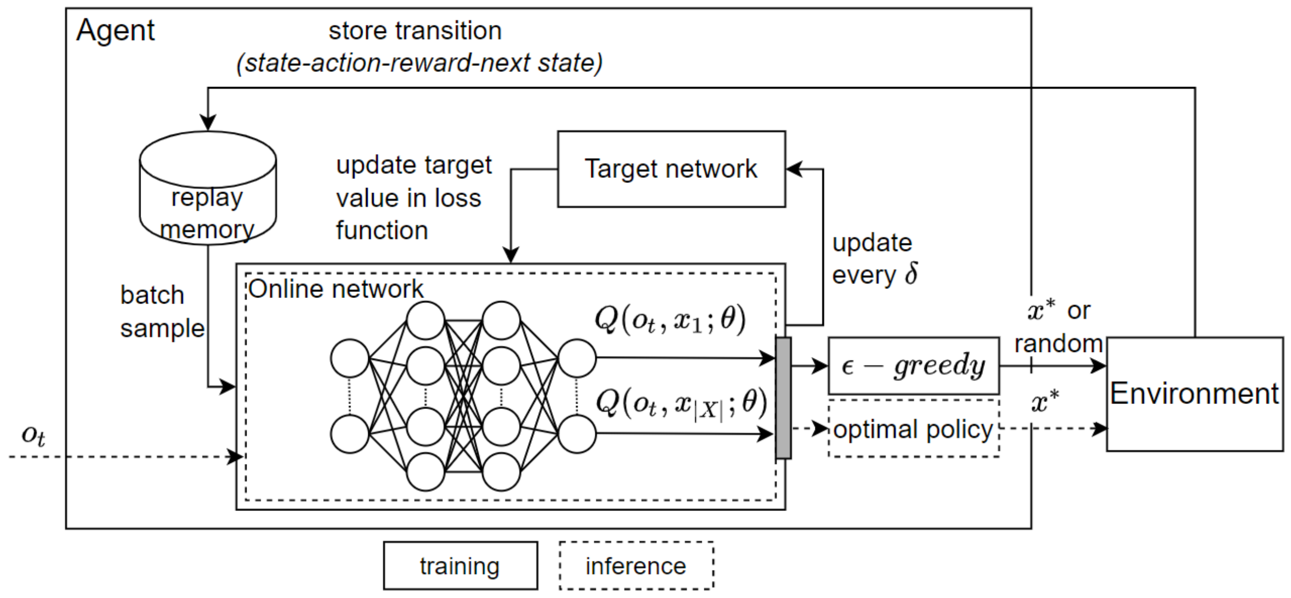


Figure 5.1: Illustration of the typical DDQL architecture.

weights and biases. However, it is only the online NN that is updated at every training step, while its weights and biases are copied to the target NN every $\delta$ training steps. The target NN (represented by $\theta^-$) is used to temporarily stabilize the target value which the online NN (represented by $\theta$) tries to predict. Effectively, the training of the learning agent boils down to the minimization of Eq. (5.3), also known as *loss function*. In the *target* term of Eq. (5.3), the greedy policy is evaluated by the online NN, whereas the greedy policy's value is evaluated by the target NN. As per [80], this reduces the overestimation of $Q$-values, which would be the case if both the greedy policy and its value have been evaluated by the online NN. An illustration of the DDQL framework is given in Fig. 5.1.

$$L(\theta_t) = \mathbb{E}\Big[\Big(\underbrace{r_t + \gamma Q\big(s_{t+1}, \underset{x \in X}{argmax} Q(s_{t+1}, x; \theta_t); \theta_t^-\big)}_{\text{target}} - \underbrace{Q(s_t, x_t; \theta_t)}_{\text{predicted}}\Big)^2\Big] \tag{5.3}$$

## 5.4 Methodology

### 5.4.1 Single-agent SFCE

We devise a single-agent RL algorithm for SFCE to serve as a baseline for comparison with our multi-agent RL scheme. Specifically, we opt for the DDQL method, as it has demonstrated promising results within our problem space (*e.g.,* [73, 74, 81]). In our implementation, a training episode consists of the placement of an entire SFC, whereas a decision step refers to the placement of a single VNF of the SFC.

**Observations.** At time $t$, the agent observes information $o_t$ about the current VNF $i$ and the SFC $G$ that it belongs to, as well as information about the available resources of the substrate physical topology $G_S$. In particular, the agent receives the length of the SFC ($|V|$), the coordinates of the *src* (*src.loc*) and *dst* (*dst.loc*) nodes of the SFC, the CPU demand of VNF $i$ ($d(i)$), the ingress ($d(i^-, i)$) and egress ($d(i, i^+)$) bandwidth requirements of VNF $i$, as well as its order (*i.order*) in the SFC. Regarding the physical network, the agent observes the coordinates (*u.loc*) and the average available CPU ($\hat{c}(u)$) of each PoP $u \in V_S$ at time $t$, and the available bandwidth ($c(u, v)$) of each link $(u, v) \in E_S$ at time $t$. That is:

$$\begin{aligned} o_t = \big( & \underbrace{|V|, src.loc, dst.loc,}_{\text{SFC state}} \\ & \underbrace{d(i), d(i^-, i), d(i, i^+), i.order,}_{\text{VNF state}} \\ & \underbrace{(u.loc, \hat{c}(u), \forall u \in V_S), (c(u, v), \forall (u, v) \in E_S)}_{\text{substrate state}} \big) \end{aligned}$$

Notice that, since $o_t$ does not hold information about the individual servers of the topology (which are the elements that actually host VNFs), the environment is partially observable, hence we use *observation* $o_t$ instead of *state* $s_t$.

**Actions.** In our centralized implementation, an action determines which PoP will host the current VNF $i$. Concretely, the agent's action set is $X = \{1, 2, ..., |V_S|\}$. Notice that, if decision steps referred to the placement of an entire SFC instead of individual VNFs, then the action space would have been substantially larger (*i.e.,* $|X| = |V_S|^{|V|}$). This would have an adverse effect on the algorithm's performance.

**Reward.** A VNF placement is deemed successful if the selected PoP has at least one server with adequate resources to host it. An SFC placement (which is the ultimate goal) is deemed successful, if all of its VNFs have been successfully placed and all of its virtual links are assigned onto physical paths that connect the VNFs correctly. For every successful VNF placement, the agent receives $r_t = 0.1$. If the current VNF is the last VNF of the chain (*i.e., terminal* VNF) and is successfully placed, then the virtual link placement commences (using Dijkstra's shortest path algorithm for every adjacent VNF pair - similar to [82, 83]). If this process is successfully completed, the reward is computed as follows:

$$r_t = 10 \cdot \frac{|opt\_path| + 1}{|act\_path| + 1} \tag{5.4}$$

where $|opt\_path|$ is the length of the shortest path between the *src.loc* and the *dst.loc* (recall that these are always associated with PoP locations), and $|act\_path|$ is the length of the actual path established by the DDQL algorithm. Apparently, the best reward $r_t = 10$ is given when $|opt\_path| = |act\_path|$. In case the placement of any VNF or virtual link fails (due to inadequate physical resources), then $r_t = -10$ and a new training episode initiates.

**Architecture.** As hinted by previous discussions, the functionality of our DDQL agent relies on four key elements, namely an online NN, a target NN, a replay memory, and an action selection strategy. Both NNs comprise an input layer whose size equals the length of the observation vector $o_t$ (*i.e.,* $|o_t| = 3|V_S| + |E_S| + 9$), two fully-connected hidden layers with 256 neurons each, and an output layer with $|V_S|$ neurons. All layers are feed-forward, the activation function applied on individual cells is Rectified Linear Unit (ReLU), and the loss function used is Eq. (5.3). The replay memory is implemented as a queue, which stores the latest $10,000$

**Algorithm 3** DDQL training procedure

```
1: Input: sfc_dataset, topology
2: env = Environment(sfc_dataset, topology)
3: agent = DDQLAgent(topology)
4: for sfc in sfc_dataset do {training episode}
5:     done = $False$
6:     state = env.reset()
7:     env.place_src_dst()
8:     score = 0
9:     while not done do {training step}
10:        action = agent.choose_action(state)
11:        new_state, reward, done = env.step(action)
12:        score += reward
13:        agent.store(state, action, reward, new_state)
14:        agent.learn() {train the online NN}
15:        state = new_state
```

environment transitions. The online NN samples 64 random transition instances out of the replay memory at every training step, while the target NN is updated every $\delta = 20$ steps. Last, we employ an $\epsilon - greedy$ action selection strategy, where, in Eq. (5.2), we set $\epsilon_0 = 1$ and $\epsilon_{decay} = 0.9998$.

**Training.** The instrumentation of the above is summarized in Algs. 3 and 4. In more detail, Alg. 3 describes the steps of the DDQL training process, which takes place over a collection of SFCs (*sfc_dataset*) and a physical PoP topology (*topology*) (**line 1**). At each episode (**line 4**), the agent handles the placement of a unique SFC. Prior to any action, the environment resets to a new state in **line 6** (*i.e.,* we increment the SFC index in the *sfc_dataset* by one, obtain the first VNF of the current SFC, and generate random loads for PoPs and physical links), and assigns the *src* and *dst* of the SFC to the respective PoPs (**line 7**). The core learning process lies within **lines 9-16**. First, the agent chooses an action based on the current state (**line 10**), the environment transitions to a new state based on the action being taken (**line 11**), the transition is stored in the replay memory of the agent (**line 13**), the online NN is trained (**line 14**), and the current score and state are updated (**lines 12** and **15**). The transitions of the environment to a new state are further described in Alg. 4, which practically implements the reward computation and episode termination logic described earlier (see the *Reward* paragraph).

**Algorithm 4** Environment transition procedure

```
1: Input: action
2: done = False
3: PoP = decode(action)
4: n_success = place_node(vnf, PoP)
5: if n_success and vnf.is_terminal then
6:    l_success, act_path, opt_path = place_links(sfc, topo)
7:    if l_success then {successful SFCE}
8:       reward = $10 \cdot \frac{|opt\_path|+1}{|act\_path|+1}$
9:    else {insufficient link capacities}
10:      reward = $-10$
11:   new_state = next_sfc()
12:   done = True
13: else if n_success then {successful VNF allocation}
14:   reward = 0.1
15:   new_state = next_vnf()
16: else {insufficient PoP capacity}
17:   reward = $-10$
18:   new_state = next_sfc()
19:   done = True
20: return new_state, reward, done
```

## 5.4.2 Multi-agent SFCE

We implement a DMARL scheme for SFCE. Specifically, we generate a single DDQL agent (as described in Section 5.4.1) for every PoP $u$ in the substrate network $G_S$. For the rest of this section, we assume that agent $\alpha$ is associated with PoP $u$, and agent $\beta$ with PoP $v$.

**Observations.** At time $t$, each agent $\alpha \in A$ observes information $o_t^\alpha$ about the current VNF $i$ and the SFC $G$ to which this VNF belongs, as well as the available resources of the substrate physical topology $G_S$. In detail, agent $\alpha$ receives the length of the SFC, the coordinates of *src* and *dst* nodes of the SFC, the CPU demand of VNF $i$, the ingress and egress bandwidth requirements of VNF $i$, and the order of this VNF in the SFC, similar to the single-agent observation mentioned in Section 5.4.1. Regarding the physical network, agent $\alpha$ observes the coordinates of PoP $u$, the available CPU ($c(u_s)$) of every server $s \in u$, and the available bandwidth of each physical link connected to PoP $u$.

We further augment the observation space of agents by enabling *cross-agent communications.* Specifically, we define $N(\alpha)$ to be the set of neighboring agents of $\alpha$, *i.e.,* $N(\alpha) = \{\beta :$

$distance(\alpha, \beta) = 1, \forall \beta \in A - \{\alpha\}\}$, where $distance(\alpha, \beta)$ refers to the length of the shortest path between PoPs $u$ and $v$ in $G_S$ (recall that $\alpha$ operates over $u$ and $\beta$ over $v$). Effectively, $\alpha$ receives the location coordinates and the average CPU capacity from every PoP $v$ managed by a neighboring agent $\beta \in N(\alpha)$. As it will become apparent, cross-agent communications are pivotal for the efficiency of the system, since they enable agents to reason about the state of other agents. Yet, to maintain the communication overhead low, agents do not share their entire local state (*i.e.,* the available CPU of each server - $c(u_s), \forall s \in u$), rather a single descriptive value (*i.e.,* the average available CPU across all servers - $\hat{c}(u)$).

As explained in Section 5.3.2, agents within MARL settings operate over non-stationary environments, which practically implies that old experiences might become highly irrelevant as agents shift from exploratory to exploitative. To this end, we include a *fingerprint* [84] into the observation, namely $\epsilon$ (the probability to select a random action), as a means to distinguish old from recent experiences.

The last elements that are inserted into $o_t^\alpha$ are three binary flags, namely $hosts\_another$, $hosts\_previous$, and $in\_shortest\_path$, which are computed by agent $\alpha$ prior to action selection. In particular, $hosts\_another = 1$ if any VNF of the current SFC has been placed in PoP $u$, $hosts\_previous = 1$ if the previous VNF of the current VNF has been placed in PoP $u$, and $in\_shortest\_path = 1$ if PoP $u$ is part of the shortest path between the *src* and the *dst* of the SFC, while they are zero otherwise. These three flags will be proven crucial for analyzing the behavior of the DMARL system in Section 5.5.3. As such, we define:

$$
\begin{aligned}
o_t^\alpha = \big(&\underbrace{|V|, src.loc, dst.loc,}_{\text{SFC state}} \\
&\underbrace{d(i), d(i^-, i), d(i, i^+), i.order,}_{\text{VNF state}} \\
&\underbrace{u.loc, (c(u_s), \forall s \in u), (c(u, v), \forall v \in N(u)), \epsilon,}_{\text{local state (resources and fingerprint)}} \\
&\underbrace{hosts\_another, hosts\_previous, in\_shortest\_path,}_{\text{local state (flags)}} \\
&\underbrace{(v.loc, \hat{c}(v), \forall v \in N(u)}_{\text{neighbors' condensed state}}\big)
\end{aligned}
$$

where $N(u)$ denotes the neighboring nodes of $u$ in the graph $G_S$. Note that, contrary to the single-agent case, here the team of agents observes the true state of the environment, *i.e.,* $\cup_{\alpha \in A} o_t^\alpha = s_t$. However, from the point of view of a single agent, the environment is still partially observable.

**Actions.** Agents are equipped with a set of actions that allows them to express their intents based on their local state. In detail, all agents share the same discrete set of actions $X = \{0, 1, 2\}$, where 0 indicates low willingness, 1 expresses a neutral position, and 2 indicates high willingness, with respect to hosting the current VNF. Note that the above action sets are topology-agnostic (in contrast to the action set of our single-agent DDQL, where there is one action for each PoP).

**Action coordination.** The joint action $\boldsymbol{x}_t = (x_t^1, ..., x_t^n)$ of all agents is forwarded to an *action coordination* module positioned at the NFVO layer. This treats individual actions as *soft* decisions and eventually computes the *firm* decision according to the overall objective (*e.g.,* load balancing, consolidation) and constraints (*e.g.,* each VNF at exactly one PoP). Here, we opt for a simple action coordination scheme where (i) the agent with the highest action (ties broken arbitrarily) will enable its associated PoP to be selected to host the current VNF, and (ii) the objective is solely dictated by the reward function, meaning that the coordinator does

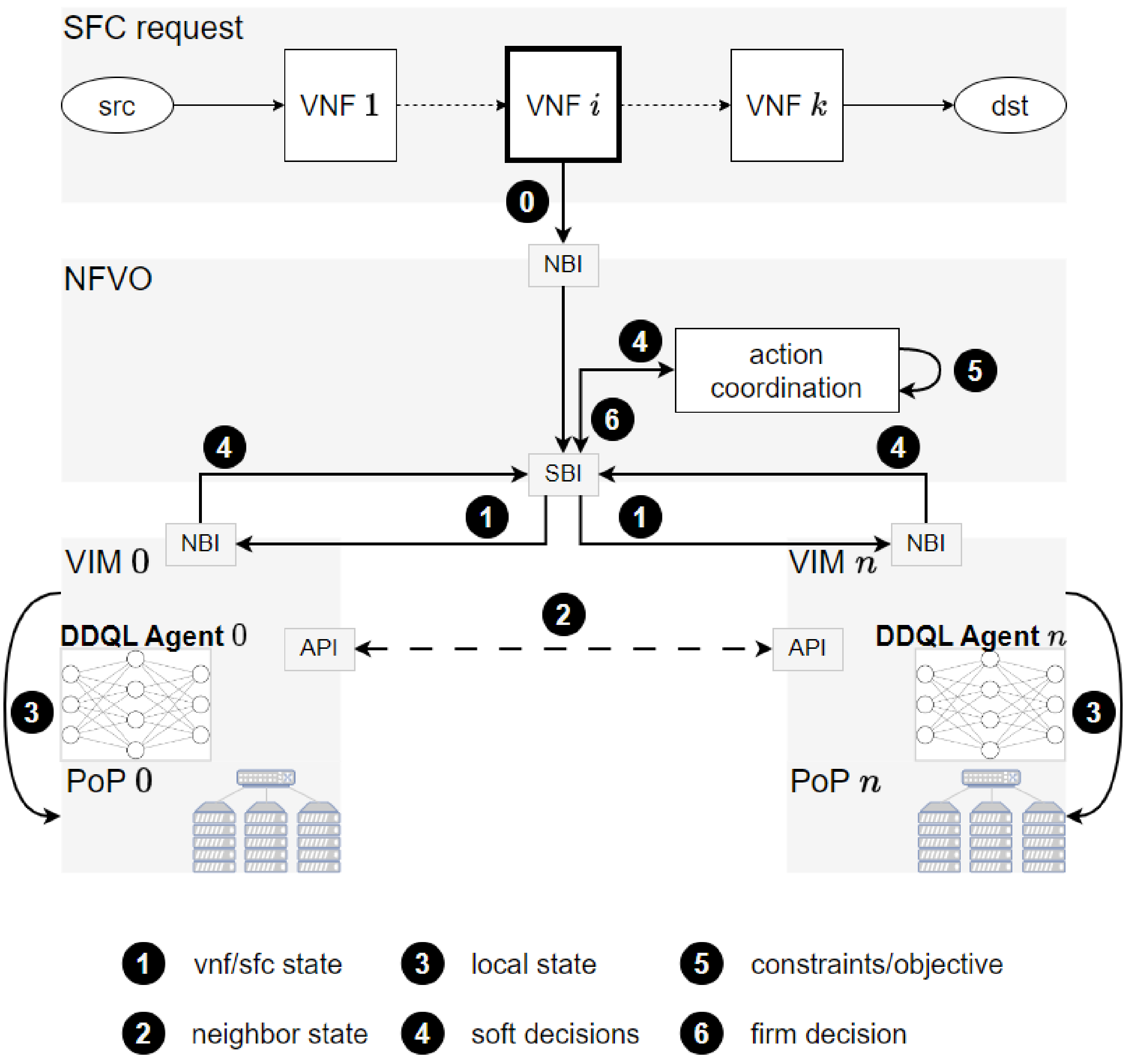


Figure 5.2: The proposed DMARL scheme is decomposed from step 0 to step 6 and mapped to the NFV MANO framework.

not intend to achieve another objective. For example, if the joint action is $\boldsymbol{x}_t = (x_t^\alpha = 2, x_t^\beta = 0)$ for the placement of a VNF $i$, the action coordination module will infer that $i$ will be positioned at node $u$, which is the PoP of agent $\alpha$, since $x_t^\alpha > x_t^\beta$.

**Reward.** Our DMARL algorithm adopts a reward scheme similar to the one presented in the single-agent case. That is, for every successful VNF placement, all agents receive $r = 0.1$, and for every successful SFC placement the common reward is computed in Eq. (5.4). In case of a failed SFCE attempt (inadequate physical resources), agents receive $r = -10$.

**Architecture.** The proposed DMARL scheme is shown in Fig. 5.2, which illustrates the mapping of the proposed elements and functionalities onto the NFV MANO framework. At

**Algorithm 5** DMARL training procedure

```
1: Input: sfc_dataset, topology, n_actions
2: env = Environment(sfc_dataset, topology)
3: marl = MARL(topology, n_actions)
4: for sfc in sfc_dataset do {training episode}
5:     done = $False$
6:     state = env.reset()
7:     env.place_src_dst()
8:     score = 0
9:     while not done do {training step}
10:        soft_actions = marl.choose_actions(state)
11:        firm_action = env.coordinate(soft_actions)
12:        new_state, reward, done = env.step(firm_action)
13:        score += reward
14:        marl.store(state, soft_actions, reward, new_state)
15:        marl.train() {train the online NNs of all agents}
16:        state = new_state
```

the top, we depict an SFC request which is conveyed to the NFVO via its northbound interface (NBI) in **step 0**. During **step 1**, the NFVO forwards the VNF and the SFC states to the underlying VIMs, and we assume that each VIM is associated with a single PoP. Additionally, each VIM is equipped with a DDQL agent similar to the one described in Section 5.4.1, their only difference being the input and output layers. In particular, the input layer of each DDQL agent $\alpha$ in the DMARL setting has $|o_t^\alpha|$ neurons, while its output layer consists of three neurons (one for each action). The API calls depicted in **step 2** implement the cross-agent communication functionality, where each agent retrieves condensed information from its neighbors. **Step 3** refers to the computation of local state, *i.e.,* monitoring of available CPU of each server and updating the three binary flag values. Afterwards, agents convey their soft actions via the NBIs of their VIMs towards the southbound interface (SBI) of the NFVO (**step 4**), where individual actions are aggregated into a joint action that is handed over to the coordination module. The latter resolves potential conflicts and, in principle, assesses the individual preferences with respect to the overall objective (**step 5**). Finally, the firm decision, which indicates the PoP that will host the current VNF, is sent to the respective VIM (**step 6**). This process is repeated until all VNFs and virtual links are assigned, or until the SFCE fails (virtual links are assigned via Dijkstra's method, similar to the single-agent case - cf. Section 5.4.1).

**Training.** The training procedure followed by the proposed DMARL scheme is outlined in

Alg. 5, which exhibits many similarities with Alg. 3. Their core differences are as follows. First, in **line 1**, the number of actions (*n_actions*) is provided as input. That is, the number of actions is no longer determined by the number of PoPs; it rather becomes a parameter of the multi-agent framework. Specifically, the number of actions affects (i) the capacity of agents to express their resource allocation intents and (ii) the duration of the system's convergence. As a good compromise, we always set the number of actions to three in this work, *i.e.,* actions are taken from the set $\{0, 1, 2\}$. Another difference between the two algorithms lies in the manner in which the *state* is interpreted in **line 6**. In Alg. 5, *state* refers to the VNF and the SFC state only, which is shared across all agents (see **step 1** in Fig. 5.2). However, in **line 10**, further subroutines are called so that each agent chooses an action based on additional observations, such as its local state and neighboring (condensed) states (cf. **steps 2** and **3** in Fig. 5.2). In **lines 11** and **12**, the coordination module derives the firm action, and the environment transitions to a new state based on this action, as explained in **steps 4** to **6** in Fig. 5.2. In **line 14**, each agent stores its transition into its replay memory (each agent expands the *state* variable), and in **line 15** all agents train their online NNs.

## 5.5 Evaluation Results

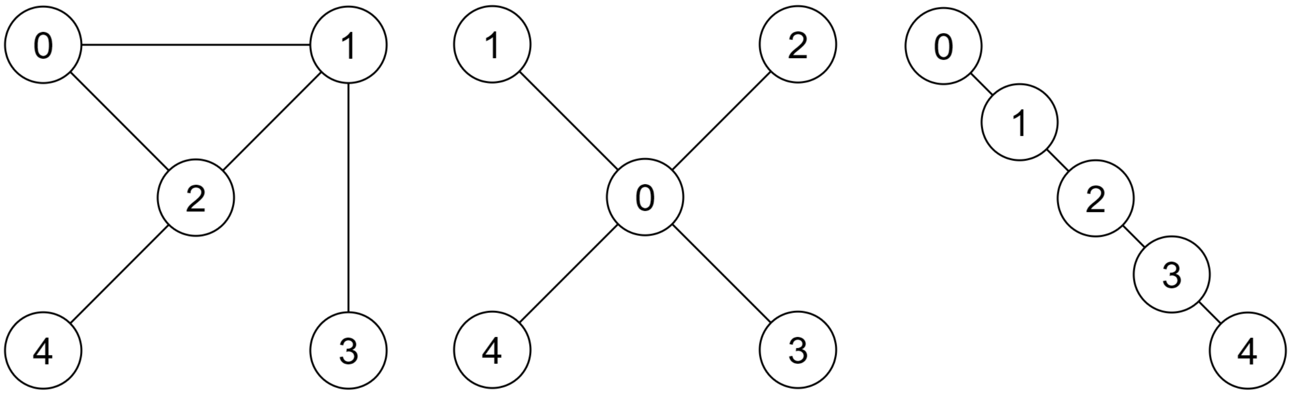
This section covers a wide range of evaluation aspects, primarily focusing on the proposed DMARL algorithm. Initially, we present the simulation settings and the main performance metrics that we take into consideration. Then, we compare the single-agent DDQL method with the DMARL scheme, and elaborate on the learning behavior developed by the team of agents. Subsequently, we evaluate the cross-agent communication feature, and measure its implications across various physical network topologies. Finally, we experiment with an imperfect cross-agent communication scheme, which is more grounded to reality.

### 5.5.1 Evaluation Setup

We consider linear SFCs consisting of two to five VNFs (excluding the *src* and *dst* auxiliary VNFs). Unless otherwise specified, the algorithms are trained with datasets containing $10,000$ SFCs (*i.e.,* $10,000$ training episodes). Each VNF of an SFC requires $5-20\%$ of a server's CPU. Each time a new SFC requests embedding, the physical resources reset to arbitrary states. In particular, all previously embedded VNFs and virtual links are removed, and all servers generate random CPU loads within $70-100\%$. That is, their available CPU lies in $0-30\%$, and, in this way, we reduce the risk of precluding feasible embeddings due to CPU insufficiency. This approach (i) renders consecutive SFC placements (*i.e.*, episodes) independent events, and (ii) enables us to reason about the actual learning efficiency of the algorithms, as low scores will be solely due to insufficient learning. Further, each PoP comprises ten servers, *i.e.,* each PoP is a (micro) datacenter. Regarding the bandwidth of inter-PoP physical links, we assume that it always suffices for virtual link allocation in our current study, and aim to explore extensive insights into the aspect of virtual node allocations.

As illustrated in Fig. 5.3, we utilize three PoP topologies (*i.e.,* $G_S$) in our evaluations, which have been selected to unveil critical properties of the multi-agent framework. In particular, each topology comprises five PoPs; hence, the (single-agent) DDQL algorithm works with five actions (one for each PoP), and the DMARL framework generates five independent DDQL agents (one per PoP). We note that the auxiliary *src* and *dst* VNFs of an SFC are associated with random PoPs (*i.e., src.loc* and *dst.loc* might even coincide; in this case, the length of the optimal path $|opt\_path|$ is zero).

While we employ several micro-benchmarks in order to interpret and assess the algorithmic behaviors, two informative metrics refer to the tracking of (i) the accumulated reward, and (ii) the rate of optimal and rejected partitionings.



(a) Mesh (b) Star (c) Linear

Figure 5.3: Topologies used for evaluation.

### 5.5.2 Comparison with state-of-the-art

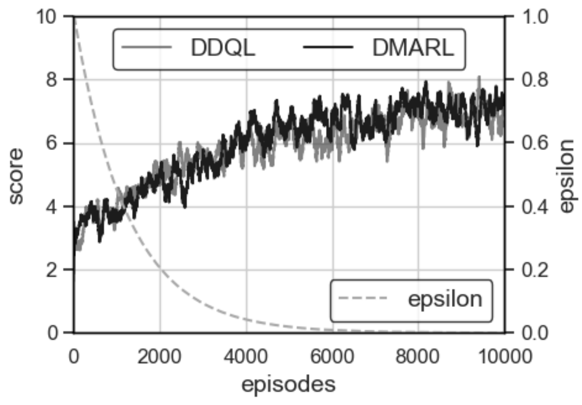
We compare the performance of the (centralized) DDQL against the (distributed) DMARL algorithm over the mesh topology shown in Fig. 5.3a. Recall that DDQL serves as our state-of-the-art benchmark method. Instead of using existing DDQL schemes for SFCE (*e.g.,* [73, 74, 81]), we devise a DDQL implementation which better matches our problem formulation and objective (cf. Section 5.4.1), thus allowing for a fair comparison.

At first glance at Fig. 5.4a, where we plot the corresponding scores as moving averages across the last 100 episodes, we observe that both methods perform similarly. In particular, towards the end of the experiment, the two algorithms manage to retrieve an average score of 7.0. Taking Eq. (5.4) into account, we infer that both learning schemes obtain SFCEs that are, on average, 30% from the optimal. However, by zooming in at the $4,000 - 6,000$ episodes interval, it becomes apparent that the DMARL framework exhibits faster convergence towards the aforementioned score than DDQL. In fact, this is further corroborated by Fig. 5.4b, where it becomes clear that, at the exact same interval, the rate of optimal embeddings of DMARL grows more rapidly than that of DDQL. According to Fig. 5.4b, the rejection rates are negligible for both algorithms.

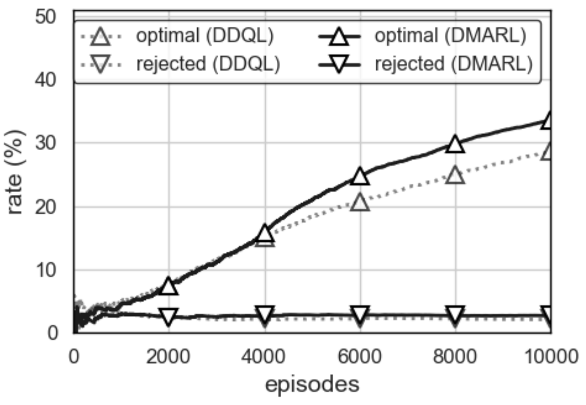
Although informative, Figs. 5.4a and 5.4b do not convey a lot about the differences of the underlying algorithmic behaviors. To this end, we also consider Fig. 5.4c, which depicts the footprint of optimal SFCEs. The most evident difference here is centered on the incapacity of

DDQL to achieve many *1 PoP* embeddings, which is counterbalanced by its higher capacity to achieve *>2 PoP* embeddings, compared to DMARL. Our interpretation of this behavior is as follows: DDQL has a partial view of the real state, but its observation contains information about the entire topology (see Section 5.4.1). This enhances its inherent capacity to distribute VNFs across many PoPs, since it can reason about the state of the entire multi-datacenter system. At the same time, its observations only include the average available CPU capacities of each PoP (*i.e.,* $d_{S,t}(u)$ in $o_t$), which limits its confidence in consolidating multiple VNFs into the same PoP. For instance, if $d_{S,t}(u) = 0.15$ for the PoP $u \in V_S$, and the current VNF $i$ demands $d_R(i) = 0.18$, then it is impossible for DDQL to be certain whether $i$ fits into a server of PoP $u$. Conversely, a DMARL agent $\alpha$ operating over $u$ has a detailed view of the available CPU resources of all servers in PoP $u$, *e.g.,* $\alpha$ observes $(d_{S,t}(u_1) = 0.10, ..., d_{S,t}(u_s) = 0.20, ..., d_{S,t}(u_{10}) = 0.15)$, and, in this case, it can be certain that server $s$ of PoP $u$ can accommodate VNF $i$.

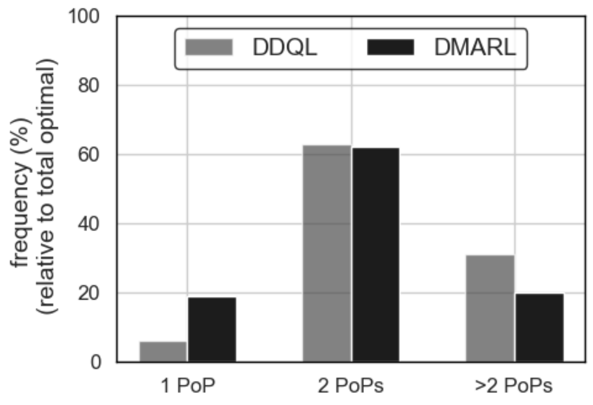
Although the aforementioned limitation can be mitigated by extending DDQL's observation space with additional metrics (at the risk of further slowing its convergence), we deem that this analysis indicates the implications of decision-making with incomplete information in the context of SFCE.



(a) Learning progress



(b) Optimal and rejected SFCs



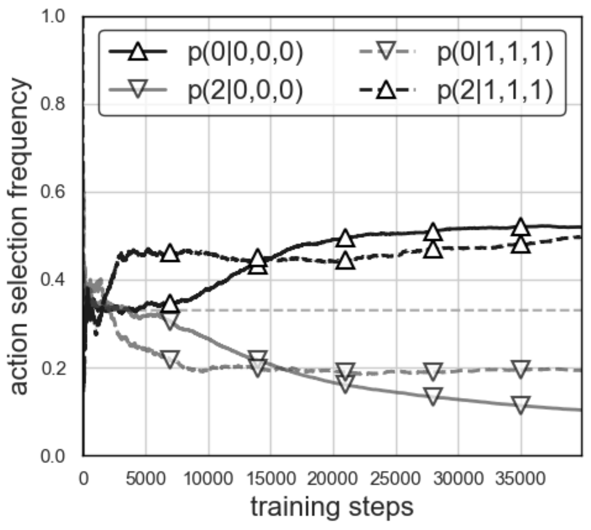
(c) SFC embedding footprint

Figure 5.4: Performance comparison between DDQL and DMARL.

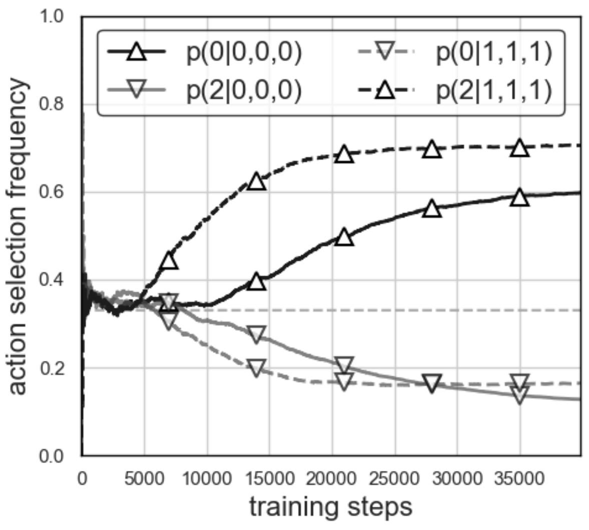
### 5.5.3 DMARL analysis

To further delve into the behavior developed by the DMARL scheme, we employ Fig. 5.5. Here, we attempt to shed light into the *learning* aspect of the team of agents in the mesh topology, by examining certain action selection *rules* that have been identified over the course of training. To this end, we monitor the evolution of $p(x|f_1, f_2, f_3)$ at every training step, which shall be interpreted as the probability of taking action $x \in X = \{0, 1, 2\}$, given that $f_1 = hosts\_another$, $f_2 = hosts\_previous$, and $f_3 = in\_shortest\_path$ (recall that these are binary flags, defined in Section 5.4.2, and are part of the observation $o_t^{\alpha}$).

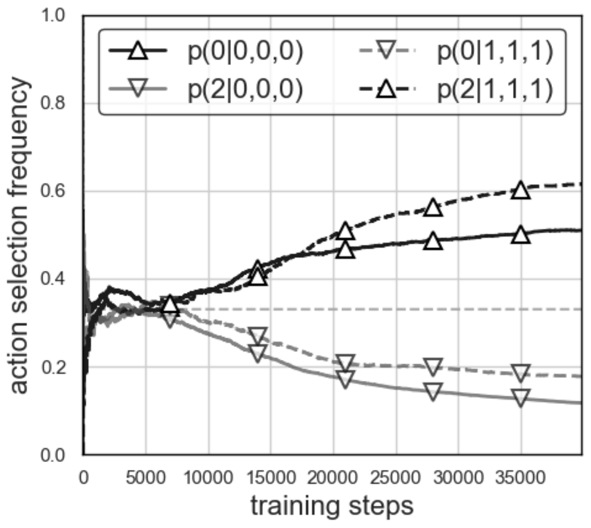
As per Figs. 5.5a - 5.5e, every agent manages to discover four key rules which, from a human perspective, seem rather intuitive for the SFCE problem. Specifically, the increase of $p(0|0, 0, 0)$ implies that the agents tend to express low willingness for hosting the current VNF when all flags are down, *i.e.,* no other VNFs are placed in the associated PoP ($hosts\_another = 0$), the previous VNF is placed in another PoP ($hosts\_previous = 0$), and the associated PoP is not in the shortest path from the *src* and the *dst* ($in\_shortest\_path = 0$). In contrast, a growing $p(2|1,1,1)$ means that agents tend to express high willingness to host the current VNF when all flags are up. At the same time, the counter-intuitive $p(2|0,0,0)$ (high willingness when all flags are down) and $p(0|1,1,1)$ (low willingness when all flags are up) seem to decrease over time. Our argument that these rules are indeed learned is backed by the fact that the respective probabilities diverge from the horizontal dotted line at $p = 0.33$, which indicates random action selection. Nevertheless, these lines never reach either 1.0 or 0.0, which may imply that the additional observation dimensions (besides the three flags) also play an important role in the action selection.



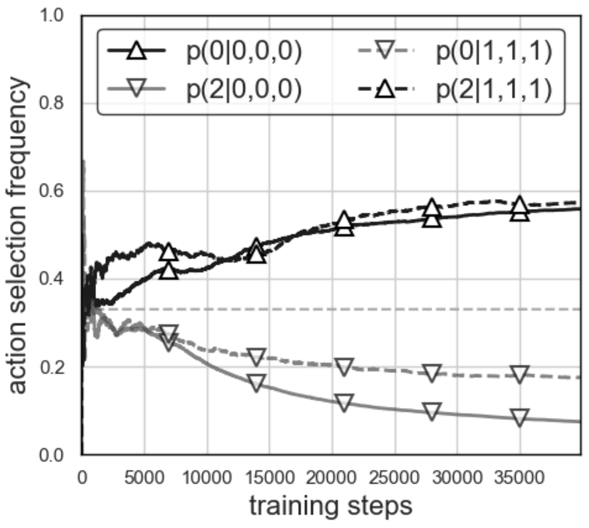
(a) Agent 0



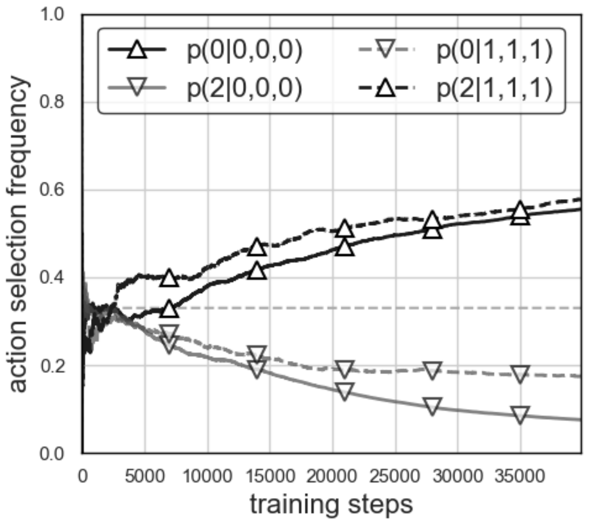
(b) Agent 1



(c) Agent 2



(d) Agent 3



(e) Agent 4

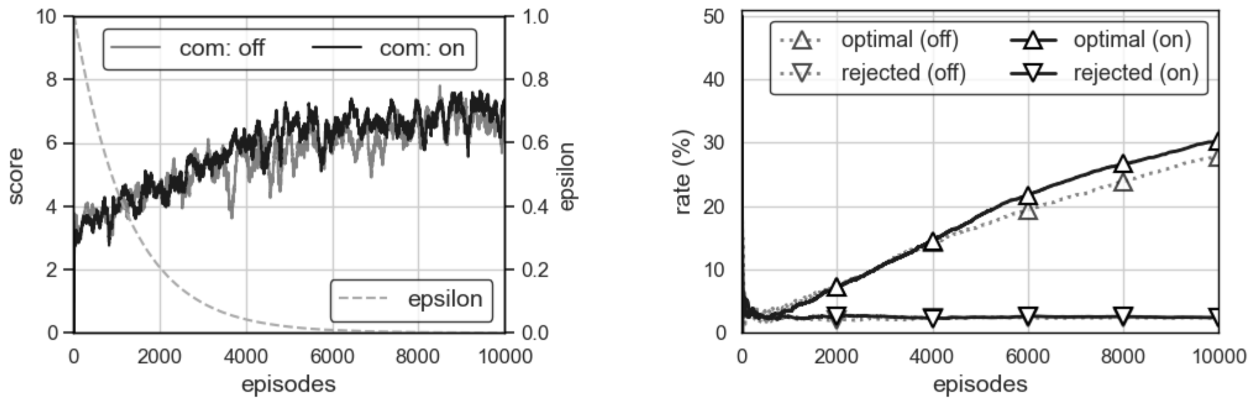
Figure 5.5: Action selection rules identified by DMARL agents.

### 5.5.4 Cross-agent communication

We now evaluate the impact of cross-agent communication on the overall efficiency of the DMARL scheme across all three PoP topologies shown in Fig. 5.3. Therefore, we implement a variant of the DMARL algorithm, where agents do not receive the neighbors' condensed state (*i.e.,* location and average available CPU in $o_t^{\alpha}$).

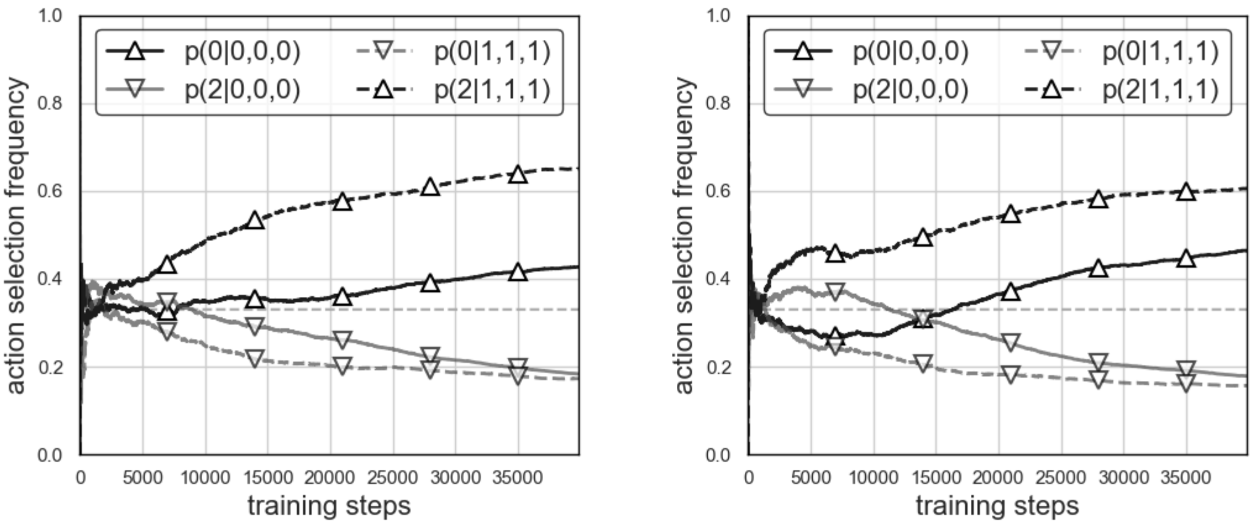
Fig. 5.6 summarizes our findings for the mesh topology (Fig. 5.3a). According to Figs. 5.6a and 5.6b, the DMARL scheme with the communication feature *on* dominates the respective scheme with communication *off*. In particular, the former manages to converge faster towards an average score of 7.0, and its rate of optimal partitionings grows slightly quicker, compared to the latter. To interpret this performance difference, we lay out Figs. 5.6c and 5.6d for DMARL with communication *off*, meant to be compared with Figs. 5.5b and 5.5c respectively, where DMARL employs cross-agent communication. We focus on *Agent 1* and *Agent 2*, since they operate over two pivotal PoPs in the mesh topology (*i.e.,* PoPs 1 and 2 are highly relevant for SFCs). From the comparison of the respective $p(x|f_1, f_2, f_3)$-lines, we observe that, when cross-agent communication is disabled, action selection *rules* are learned both slower and less firmly.

Results from the respective comparison in the star topology (Fig. 5.3b) are depicted in Fig. 5.7. In particular, Figs. 5.7a and 5.7b emphasize the superiority of the DMARL algorithm that uses cross-agent communication. Here, the former converges both faster and to a higher average reward, and the difference in optimal partitionings is close to 10%. It is worth noting that, we observed slower and less firm identification of action selection *rules* in this case also, but the respective results are omitted for brevity. Instead, we plot Fig. 5.7c, where we zoom into a behavioral aspect of *Agent 0* (since PoP 0 is a pivotal PoP in the star topology). Specifically, $p(0|0, 0, 1)$ is lower for *Agent 0* when cross-agent communication is enabled, meaning that this agent is still more likely to select actions 1 or 2 when it is in the shortest path from the *src* to the *dst*, even if it does not host another VNF or the previous VNF. Note that the respective line for DMARL with communication *off* is closer to random. This corroborates the fact that enabled communication augments *Agent 0* to perceive its position in the star topology and,



(a) Learning progress

(b) Optimal and rejected SFCs



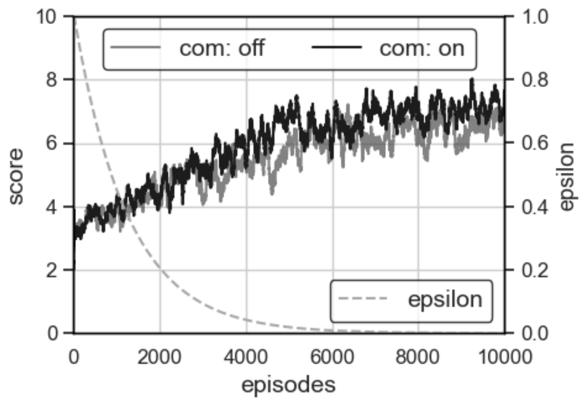
(c) Agent 1

(d) Agent 2

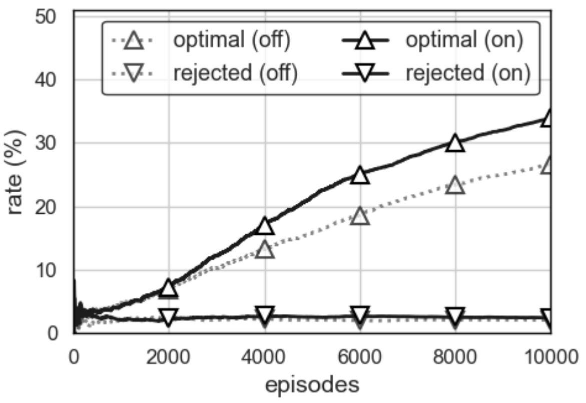
Figure 5.6: Cross-agent communication in the mesh topology.

hence, to potentially facilitate SFCEs as an intermediate PoP.

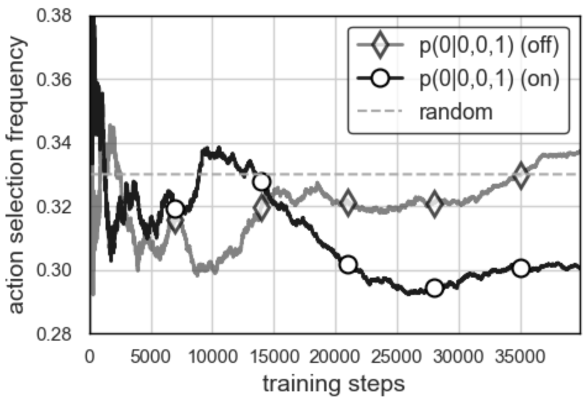
Finally, we assess the cross-agent communication feature in the linear PoP topology shown in Fig. 5.3c. Here, results regarding the learning progress (Fig. 5.8a) and the rate of optimal and rejected SFCEs (Fig. 5.8b) follow a similar trend to that of the star topology, and the benefits of cross-agent communication remain. Some insightful micro-benchmarks regarding the difference of the two schemes are illustrated in Figs. 5.8c and 5.8d. Specifically, these bar charts indicate action selection frequencies per agent during the last $2,000$ episodes (where agents are essentially purely exploitative, as $\epsilon \to 0^+$). Based on Fig. 5.8c, *Agent 1* shows the most willingness to host VNFs; however, this is contradicting to the linear topology in which



(a) Learning progress



(b) Optimal and rejected SFCs

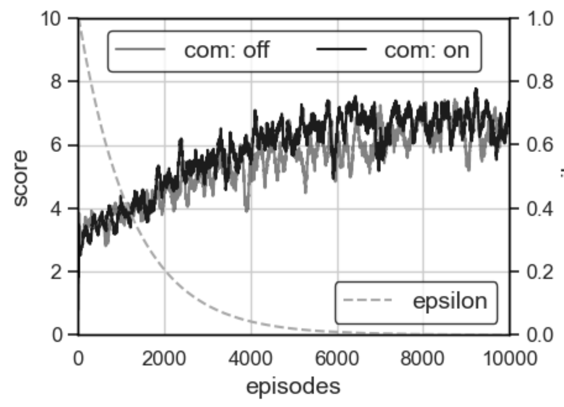


(c) Behavior of Agents 0

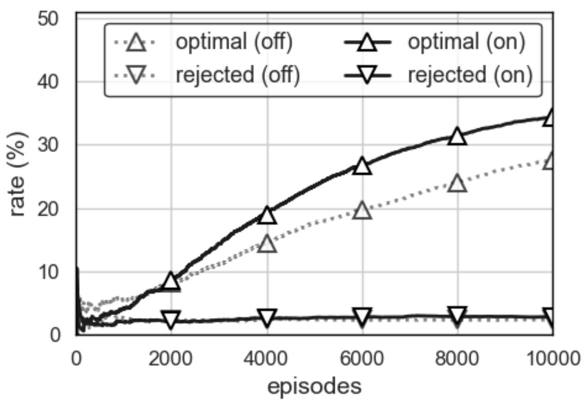
Figure 5.7: Cross-agent communication in the star topology.

we would expect *Agent 2* to participate in more SFCEs. Therefore, *Agent 2* cannot perceive its pivotal position in the linear topology when the communication is *off*. In contrast, in Fig. 5.8d, notice that *action 2* frequencies are better aligned with the position of agents in the linear topology. Specifically, we observe that *Agent 2* is the most willing to allocate its PoP resources to VNFs, followed by its adjacent agents *1* and *3*, while agents *0* and *4* participate in less SFCEs, given their outermost position in the linear topology. We also note that, in principle, when communication is disabled, agents tend to select *action 1* (*i.e.,* neutral position with respect to hosting a VNF) more frequently, which can be interpreted as lack of confidence in their learning capacity.

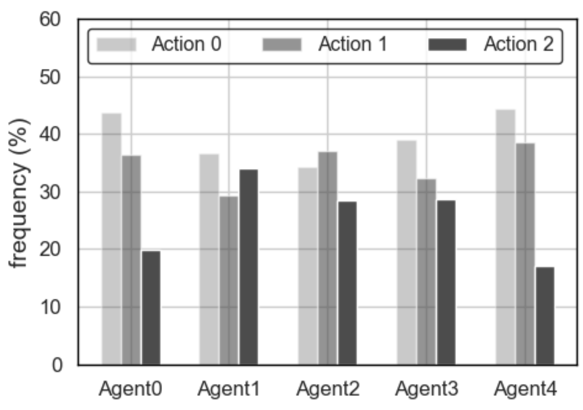
According to the analysis above, cross-agent communication of short, yet informative, messages (i) enables faster and firmer identification of action selection rules, and (ii) enhances the agents'



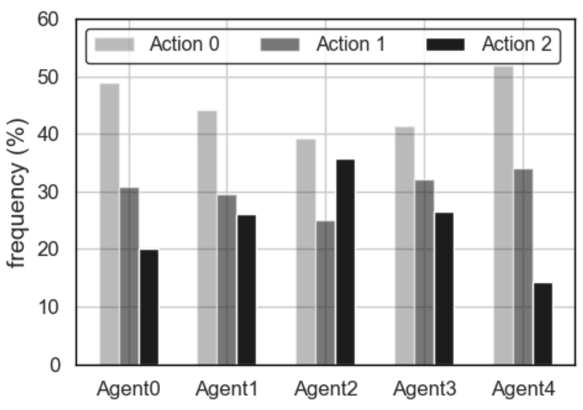
(a) Learning progress



(b) Optimal and rejected SFCs



(c) Action selection (com. off)



(d) Action selection (com. on)

Figure 5.8: Cross-agent communication in the linear topology.

ability to identify their position within the topology and adjust their action selection behavior accordingly.

### 5.5.5 Imperfect cross-agent communication

The proposed DMARL scheme relies on the exchange of concise information among neighboring agents. Effectively, this is realized via cross-VIM/PoP communications (cf. **step 2** in Fig. 5.2). However, communications are hardly ever perfect. For example, data communication may be delayed, packets may be lost, or erroneous data may appear owing to lossy (de)compression or channel fluctuation. Moreover, the communication medium may not be dedicated for a single service; therefore, the opportunities for information exchange between neighboring agents will be limited. Yet, if cross-PoP communications are totally infeasible, one can resort to the

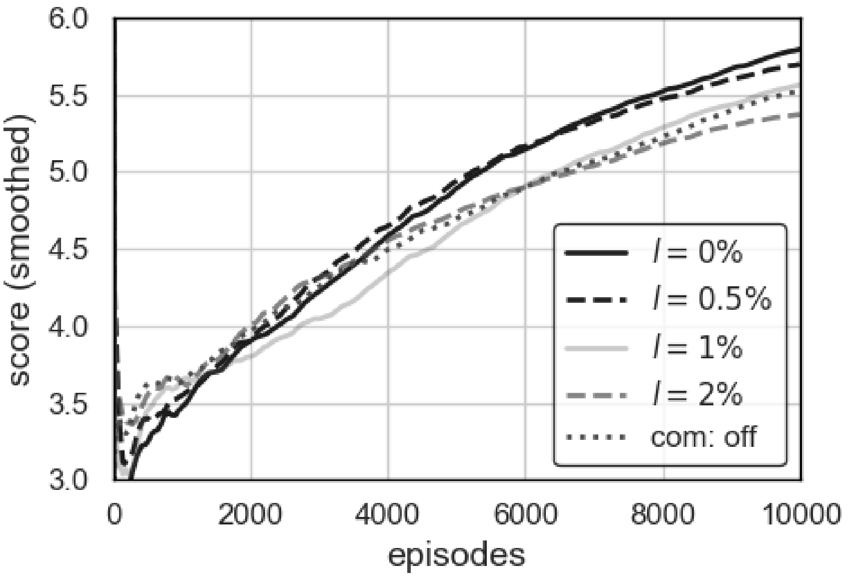


Figure 5.9: Imperfect cross-agent communication schemes.

DMARL variant with the cross-agent communication feature *off*, whose behavior has been analyzed in Section 5.5.4. Despite of some performance degradation, it still manages to optimize SFCEs to a certain degree (cf. Figs. 5.3a, 5.3b, and 5.3c).

To quantify the importance level of cross-agent communications, we evaluate our DMARL framework over lossy inter-PoP links[4] that exhibit non-zero packet loss rates $l$, (*i.e.,* $l > 0$). That is, an agent $\alpha$ receives nothing from its neighbor agent $\beta \in N(\alpha)$ with probability $l$ (in this case, the respective entries in $o_t^\alpha$ are filled with 0s), while it receives the actual condensed state of agent $\beta$ with probability $1 - l$. Here, we experiment with three levels of loss rate, *i.e.,* $l = 0.5\%$, $l = 1\%$, and $l = 2\%$, over the mesh topology (Fig. 5.3).

In Fig. 5.9, to better identify their performance differences, we plot the respective scores as moving averages across the entire training period. The observations that can be drawn are as follows. First, smaller loss rates come with better DMARL performance, while even $l = 2\%$ loss rate leads to half unit lower average score compared to the perfect communication scheme (*i.e.,* where $l = 0\%$). Another interesting outcome concerns the comparison of the imperfect communication schemes with the communication *off* scheme. As per Fig. 5.9, disabling cross-agent communication (*i.e., com: off*) results in similar scores with the DMARL which operates

[4]We here focus on the loss of neighbors' state information, rather than the packet loss of VNF data traffic.

over a mesh topology with $l = 1\%$ loss rate, and even outperforms the scenario with $l = 2\%$. That is because in the communication *off* scheme, agents learn optimized policies without ever expecting information from their neighbors. Conversely, in the communication *on* schemes, agent policies rely on cross-agent communications, and if the latter are faulty, this has an adverse effect on action selection. Further, the observation dimensions in communication *off* schemes are always less than the corresponding dimensions of communication *on* schemes. In this sense, communication *off* is not equivalent to communication *on* with $l = 100\%$.

At first glance, the above results suggest that the proposed DMARL scheme is sensitive to the loss of cross-agent communications and would require ultra-reliable communications of neighbors' states. However, this is not fully accurate. In fact, a variant of the above experiment - where each agent knows perfectly the location of its neighbors ($i.e., v.loc, \forall v \in N(u)$ for PoP $u$), and the loss rate is only applied to the neighbors' average available CPU ($i.e., d_{S,t}(v), \forall v \in N(u)$ for PoP $u$) - is conducted and only marginal differences are observed in terms of score compared to the case with ideal cross-agent communication. In conclusion, it is crucial for DMARL agents to be aware of the positions of their neighbors, as this position awareness allows them to perceive their own position within the topology more accurately and, in extension, to develop appropriate action selection behaviors. This aspect, *i.e.,* the significance of the content of the exchanged information, opens up the opportunity to apply different quality of service (QoS) requirements to different information of neighbors' states.

## 5.6 Related Work

### 5.6.1 RL in single-domain SFCE

Most studies that tackle the SFCE problem in single-domain settings opt for centralized solutions. In our context, these are centralized RL agents handling decision-making over the entire set of PoPs across the domain.

In particular, *Quang et al.* [82] devise a deep deterministic policy gradient method based on the actor-critic RL paradigm. In order to enhance the exploration capacity of their method, authors adopt a *multiple critic networks* approach. These networks are intended to evaluate multiple noisy actions produced on the basis of the *promo* action selected by the actor network. At each time step, the true action to be selected is the one with the largest mean $Q$-value, computed across all critic networks. However, the actual updates on the actor network are driven by the critic network that exhibited the lowest loss. It is worth noting that the above RL scheme merely computes placement priorities, and not actual VNF allocations. The latter are handled by a heuristic termed as *heuristic fitting algorithm*, which maps VNFs onto physical nodes in a greedy fashion based on rankings and, subsequently, utilizes Dijkstra's method to compute link assignments.

*Pei et al.* [73] propose an SFCE framework which leverages DDQL. Here, the decision-making process is subdivided into three steps. First, given a network state, a preliminary evaluation of each action is performed, *i.e.,* computation of respective $Q$-values. Out of all actions, all but the $k$ best are discarded. In the second step, these $k$ actions are executed in simulation-mode, and the actual reward and next states are observed. Finally, the action with the highest reward is performed on the physical infrastructure.

*Zheng et al.* [85] investigate the deployment of SFCs in the context of cellular core. To this end, authors put both the intra-datacenter and the intra-server placement problem into the frame. Focusing our discussion on the intra-datacenter placement, authors devise an approximation algorithm for the optimized SFCE, under the assumption that the resource requirements and the lifespan of each SFC are fully disclosed. However, given the strictness of this hypothesis, authors implement an SFCE scheme that can cope with demand uncertainty, as well. In particular, the proposed method is conventional $Q$-learning. Here, a state $s_t$ is expressed as the SFC type that needs to be deployed at time $t$, while an action $x_t$ represents the selection of a server. Nonetheless, conventional $Q$-learning is not a viable option when the state-action space is vast.

*Wang et al.* [74] and *Jia et al.* [86] investigate SFCE through the lens of fault-tolerance, essentially considering the deployment of redundant VNFs or entire SFCs which shall be engaged in the event of malfunction in the running SFC. In particular, *Wang et al.* devise a DDQL algorithm that is trained to compute an ordered pair $(u, v)$ where $u$ is the index of the DC for proper deployment, and $v$ the backup DC where the standby SFC instance will be deployed. Effectively, this implies that the entire SFCs will be placed in a single PoP. *Jia et al.* decompose the SFC scheduling problem with reliability constraints into two sub-problems. First, they use a heuristic to determine the number of redundant VNF instances (per VNF). Then, they employ an A3C algorithm which, in conjunction with a rule-based node selection approach, facilitates the process of mapping these instances onto compute nodes. In practice, the DRL algorithm learns whether to defer or not the deployment of a VNF, instead of learning where to map it. However, given such a topology-independent action space, the proposed DRL scheme is not sensitive to evolving topologies.

In contrast to the above, our work promotes single-domain SFCE in a decentralized, yet coordinated fashion, addressing various limitations of existing studies. First, the proposed DMARL scheme empowers local controllers (*i.e.,* VIMs) with the ability to express their own resource utilization intents based on detailed local observations (as opposed to *e.g.,* [87, 88, 82, 73, 85, 89, 83, 74, 86]). This is explicitly achieved by designating one DDQL agent per VIM, and by utilizing action sets that express the degree of willingness with respect to hosting VNFs. Second, our DMARL scheme can indeed claim decision-making with complete information, since, from the perspective of the team of agents, the environment is fully observable. This is not the case with many studies, which either define incomplete representations of the true state or assume very detailed knowledge at the orchestration layer (*e.g.,* [87, 82, 73, 85, 83, 74]). Third, the action spaces used by individual DDQL agents of the DMARL framework are topology-agnostic (as opposed to *e.g.,* [87, 82, 88, 73, 85, 89, 83, 74]), which alleviates scalability limitations and renders our framework a promising candidate for dynamic topologies.

### 5.6.2 RL in multi-domain SFCE

Multi-domain SFCE is commonly solved via distributed methods [90]. This can be attributed to the limited information sharing among different providers, which hinders the possibility of solving SFCE in a centralized fashion.

In [91], authors assign a DDPG-based RL agent to each domain. Further, the SFC request is encoded by the client (*i.e.,* the entity that wants to deploy the SFC) and conveyed to each RL agent. The latter then treat the SFC encoding as state and compute an action, which is effectively the bidding price for renting out their resources to the SFC request. Prices are accumulated by the client, who opts for the best combination based on a cost-based first fit heuristic. The MARL setting here is inherently competing, as agents do not have any incentive to maximize the rewards of other agents.

*Toumi et al.* [81], propose a hierarchical MARL scheme which employs a centralized agent (*i.e.,* multi-domain orchestrator - MDO) on top of multiple local-domain agents, where all agents are implemented with the DDQL architecture. The MDO receives the SFC request, and decides which local domain will host each constituent VNF. Afterwards, the agents of local domains are responsible for placing the sub-SFCs within their own nodes. Admittedly, the proposed MDO is quite similar to our own DDQL implementation for SFCE. If we assume that the partial observation of our DDQL due to data aggregation is equivalent to the partial observation of the MDO due to limited information sharing across multiple domains, then we can think of the comparison in Section 5.5.2 as a comparison of DMARL with MDO[5].

In [92], *Zhu et al.* devise a MARL scheme for SFCE over IoT networks. Specifically, they propose a hybrid architecture that employs both centralized training and distributed execution strategies. Here, the SFCE problem is modeled as a multi-user competition game model (*i.e.,* Markov game) to account for users' competitive behavior. In their implementation, the centralized controller performs global information statistical learning, while each user deploys service chains in a distributed manner, guided by a critic network.

While multi-domain SFCE is vertical to single-domain SFCE on the grounds of information disclosure, the related solutions exhibit some similarities to the proposed DMARL scheme. In effect, agents voting on hosting VNFs can be paralleled to agents bidding for resources,

[5]We acknowledge that these two sources of partial observability are not equivalent, hence a direct comparison of DMARL with MDO makes little sense.

while our coordination module can be seen as a centralized broker which aggregates votes and computes the best action. However, the intrinsic competition that underlies multi-domain scenarios cannot be overlooked. For example, notice how agents in multi-domain SFCE strive to maximize their own rewards, disregarding the rewards of others. As such, our DMARL algorithm cannot be directly compared to any solution pertaining to the above.

## 5.7 Conclusions

Both IQL and our evaluation methodology enable us to obtain valuable knowledge. Specifically, the comparison in Section 5.5.2 indicates that DMARL outperforms a centralized state-of-the-art SFCE method based on DDQL, which we mainly attribute to the full observability of the former against the partial observability of the latter. Concretely, DDQL fails to consolidate VNFs to the extend DMARL does, given the fact that it is not aware of the exact server capacities. We further corroborate that the team of agents manages to recognize certain intuitive action selection rules in Section 5.5.3. This is highly crucial, especially considering efforts towards machine-learning explainability. Moreover, we identify that disabling the exchange of concise messages among neighboring agents limits their ability to perceive their position within the multi-PoP network, which has a negative effect on the developed action selection behaviors (cf. Section 5.5.4). Last, we quantify the performance drop of DMARL over multi-PoP networks which exhibit non-zero packet loss rates (cf. Section 5.5.5). The key takeaway here is that it is highly important for DMARL agents to at least be aware of the position of their neighbors, which implies that location coordinates play an important role in the individual action selection policies.

Our work exhibits a few shortcomings, which we deem important to discuss and clarify. With respect to the underlying algorithmic framework we opted for, which is IQL, we have already acknowledged its lack of convergence guarantees. The natural next step is to examine a DMARL algorithm for SFCE based on *centralized training - decentralized execution* schemes, such as the ones proposed in [77] and [78], where finding the optimal joint policy is theoretically guaranteed (under certain assumptions). Additional limitations which we identify mostly pertain to the analysis of our method, rather on the method itself. In particular, we evaluate DMARL over relatively small multi-PoP topologies. Yet, the intention of this work is to explore in-depth the learning behavior that is developed by a team of independent RL agents in order to cooperatively solve SFCE. To this end, keeping the topologies small, not only allows us to delve easier into the behavior of individual agents, but to demonstrate the respective findings as well (*e.g.,* Fig. 5.5).

While this work sheds light on numerous fundamental aspects of distributed resource allocation within NFV, at the same time it paves the way for several new research directions. For instance, having established the perks of decentralized SFCE, a natural next step is to examine a multi-agent RL algorithm at which agents shall learn *what*, *when*, *to whom* and *how* to communicate; in other words, agents shall learn their own cross-agent communication protocol.

# Chapter 6

# Service Function Chain Graph Transformations

In the previous chapters, we have established that NFV is heavily reliant upon resource assignment mechanisms, purposed to efficiently allocate physical resources to virtual entities, such as SFCs. Typically, the latter are supplied to resource schedulers of virtual infrastructure managers, which are responsible for the computation of optimized embeddings. Arguably, most existing studies either treat SFC requests as rigid constructs, or apply optimizations on the SFC structure with no regard to the substrate network. However, in the event of high utilization or fragmentation of the available physical resources, this might narrow down, or even eliminate, the feasible solution space of the SFCE problem.

Along these lines, we explore the potential of transforming entire SFC graphs prior to their embedding, ultimately augmenting resource schedulers to obtain an optimized embedding solution. The overall transformation procedure is primarily driven by the resource utilization state of the underlying network, along with the notions of the *embedding flexibility* and the *transformation complexity*. Formally, we model the SFC transformation problem as a mixed integer linear program (MILP), and analyze its core properties in detail. Extensive simulations corroborate the feasibility of our approach, as well as the notable resource efficiency gains that can be achieved.

## 6.1 Motivation

Regardless of the nature of SFC graphs, which inevitably implies high adaptability with respect to their actual deployment, most studies handle SFCs as fixed constructs. Few exceptions to this are works that attempt to re-organize the VNF sequence so that either bandwidth requirements are minimized [43] or performance is improved, *e.g.,* via VNF parallelism [93]. In addition to these, [94] examines the potential of creating replicas of the original SFC and placing them into distinct network locations, while [95] employs a sizing step to determine the number of instances required for each VNF according to projected loads. However, to the best of our knowledge, no study applies optimization on the SFC graph considering the resource utilization state of the underlying network, which is somewhat restricting, given that SFCE consists a fundamental NFV-RA challenge.

To substantiate that, we employ a simplified example of a VNF embedding task, as shown in Fig. 6.1. Here, we assume that a virtualized infrastructure is requested to embed *VNF X*, which requires 30% of a server's CPU. On the right-hand side of the figure, we depict the servers comprising the infrastructure, along with their available CPU resources. Apparently, there is only one possible solution for the aforementioned task, which is assigning *VNF X* to *Server 2*. Yet, this will have an adverse effect on the resource utilization state of the network, since the remaining CPU will be highly fragmented, ultimately inhibiting - or even prohibiting - the realization of future embedding requests. Now notice that, if we simply split *VNF X* into two instances, each requiring half of the original CPU, it turns out that the feasible assignments rise to seven (excluding symmetrical mappings). In fact, several of these assignments can even reduce the existing CPU fragmentation issue (*e.g.*, $VNF\ X_1 \rightarrow Server\ 1$ and $VNF\ X_2 \rightarrow Server\ 3$). Conversely, there would be no particular reason for decomposing *VNF X* if, *e.g.*, all servers were unutilized.

Expanding this notion to entire VNF chains, we stress on the need for optimizing SFC requests, as these are originally supplied by service tenants, by primarily taking into account the resource utilization conditions of the substrate network. In particular, our work focuses on *determining the optimal number of instances for VNFs across entire SFCs*, in an attempt to augment

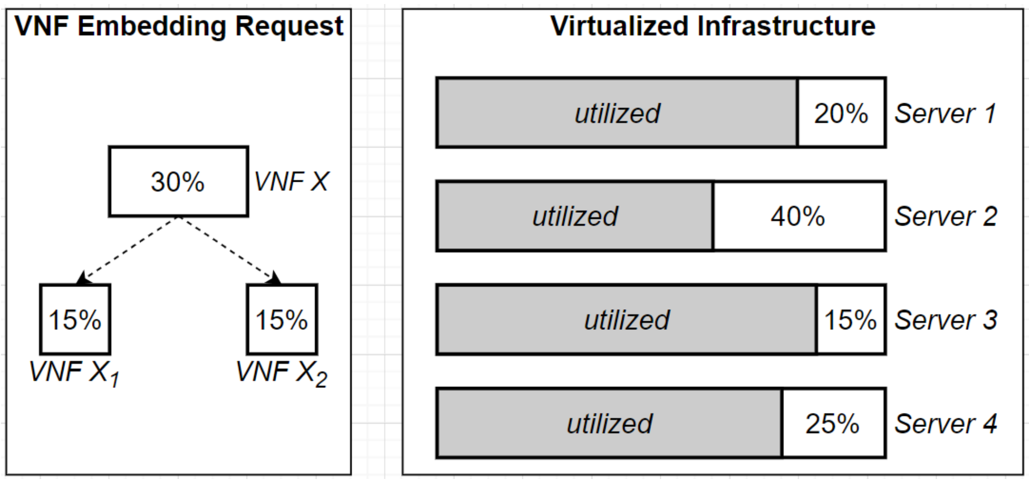


Figure 6.1: The decomposition of *VNF X* into two smaller instances expands the embedding solution space significantly.

resource schedulers obtain improved embedding decisions. We refer to this process as *SFC graph transformation (SFC-GT)*.

## 6.2 Contributions

In the context of SFC-GT, we identify two critical concepts; namely, the *embedding flexibility* and the *transformation complexity*. The embedding flexibility of a graph $G$ expresses its inherent capacity to be easily embedded in a physical topology, and is mainly captured by the degree of decomposition that $G$ has been subjected to. In contrast, the transformation complexity encapsulates practical complications that are introduced by SFC-GT. For instance, decomposing a VNF into many instances requires load balancing to correctly forward traffic among them. While we will explore both of them in detail later in this chapter, it is worth to note that the quintessence of SFC-GT boils down to a trade-off between embedding flexibility and transformation complexity, where the balancing of these two is determined by the resource utilization conditions of the underlying network. Our primary goal is to tackle this exact challenge, which, among others, involves the analysis and coherent positioning of new terms, careful mathematical representations, as well as the development of appropriate and efficient algorithmic solutions. To achieve these:

- We lay the groundwork for SFC-GT, by exploring and defining the notion of embedding flexibility and transformation complexity, practically taking a new perspective on preprocessing SFCs.
- We model the SFC-GT problem as a multi-objective MILP, whose core properties are analyzed in depth.
- We conduct extensive simulations to quantify both the resource utilization gains, as well as the overheads that are attributed to SFC-GT.

## 6.3 Problem Description

SFC-GT alleviates the inefficiency of SFCE methods at conditions that restrict the embedding solution space, such as high utilization and/or high degree of resource fragmentation across the infrastructure. Under such conditions, an SFCE solver is restricted to a small set of feasible solutions, one of which has to be picked, although it may not comply with the embedding objective. Regardless of the degree of sophistication exhibited by the embedding method, there is very little that the solver can do in such a case.

SFC-GT constitutes a viable approach to this problem. In particular, SFC-GT aspires to expand SFC graphs by decomposing VNFs into multiple instances. Each instance in the extended graph has a lower resource demand than the corresponding VNF in the initial graph. The resource demands of all instances of a specific VNF should sum up to the demand of the corresponding VNF (as expressed in the initial SFC graph). In our SFC-GT problem formulation, we discuss in detail all requirements for the extended graph. The main intuition behind the VNF instance decomposition is that VNF instances with lower resource demands can be more easily accommodated into a highly utilized or highly fragmented virtualized infrastructure, such as datacenters. Fig. 6.2 illustrates a simple example of an SFC-GT. In particular, an initial SFC graph comprising three VNFs is transformed into an extended SFC graph, at which *VNF 1* has been decomposed into three instances, whereas *VNF 2* and *VNF 3* encompass two instances, each. The extended graph also contains additional edges that connect the instances of adjacent VNFs. Note that Fig. 6.2 merely provides a simplified view of the extended graph (*i.e.,* we have omitted additional nodes inserted before decomposed VNFs for traffic distribution among the VNF instances). We discuss in full detail all attributes of the SFC extended graph in Section 6.4.1.

Since *embedding flexibility* increases with the number of VNF instances, one may expect that the SFC-GT should favor extended graphs with a multitude of instances. However, this is not the case, since additional VNF instances generate significant resource overheads that can outweigh the gains from the increased embedding flexibility. These overheads stem from various resource consumption aspects, such as the resources allocated for additional VNFs for traffic distribution among the VNF instances, switch TCAM consumption by the forwarding entries required by the additional edges in the extended graph, and the server memory consumption due to VNF state replication (we assume that VNF state is replicated among all running instances). We use the term *transformation complexity*, or simply *complexity*, to accumulate these resource overheads, which we model in Section 6.4.4. Apparently, embedding flexibility and transformation complexity are contradicting factors that both require optimization by SFC-GT. As such, the SFC-GT problem is far from trivial and requires careful attention in the modelling and balancing of these factors.

In Fig. 6.2, we depict an example of the application of SFC-GT in the context of NFV MANO frameworks. An initial SFC request is supplied to the NFVO, which is in turn conveyed to an SFC-GT module that computes an optimized transformation of the SFC in the form of an extended SFC graph. The latter is sent to the VIM for its embedding and deployment on the virtualized infrastructure. Apparently, as shown in Fig. 6.2, SFC-GT is executed prior to the SFCE.

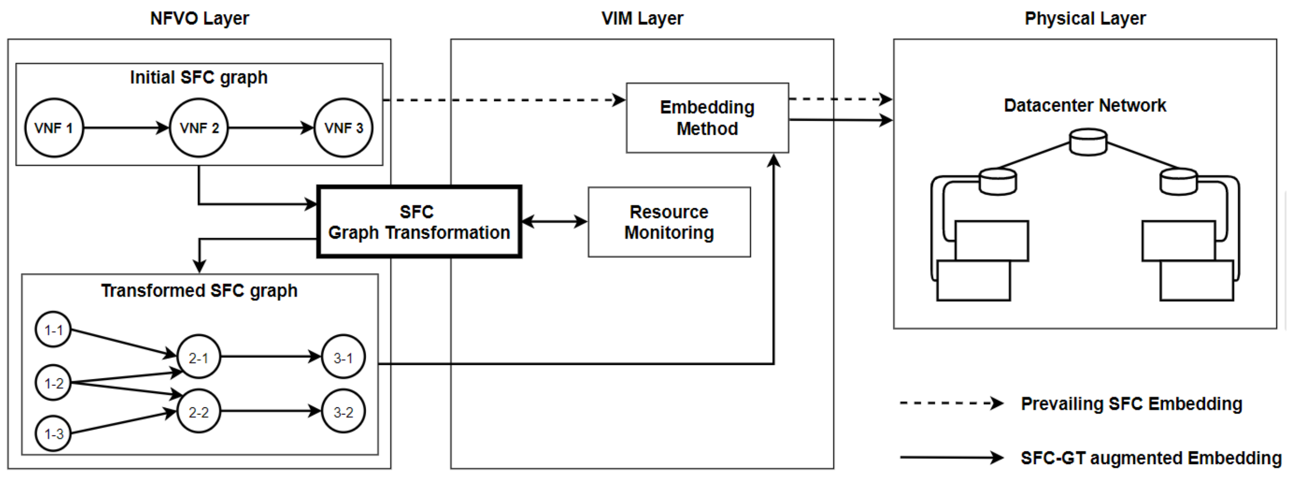


Figure 6.2: *SFC-GT augmented* embedding approach.

## 6.4 Methodology

### 6.4.1 Extended graph

Recall that we model an SFC with a directed graph $G = (V, E)$, where $V$ and $E$ denote the set of nodes and edges (*i.e.,* VNFs and virtual links, respectively). Additionally, let $d(i)$ express the CPU demand and $d^i$ the inbound traffic of VNF $i \in V$.

**Extended Service Function Chain (E-SFC) model.** The E-SFC directed graph, denoted by $G^e = (V^e, E^e)$, is linked to a graph $G$ that represents an arbitrary SFC. Effectively, $G^e$ comprises all nodes and edges that could participate in a transformation of $G$. That is, we generate several instances for each VNF, based on pre-defined values. Load balancing (LB) nodes, along with respective edges, are inserted into $G^e$ to ensure correctness of potential connections among VNF instances.

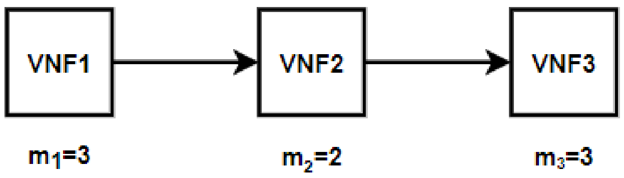
*Virtual nodes of $G^e$.* Given an SFC $G$ comprising $k$ VNFs ($k \in \mathbb{Z}^+$), we define $\mathbf{m} = [m_1, ..., m_k]$, where $m_i$ denotes the maximum number of available licenses for VNF $i$. In addition, we use $v_i$ to express the LB placed before VNF $i$ ($i \in K = \{1, ..., k\}$), $v_{i,j}$ is the $j^{th}$ instance of VNF $i$, ($j \in K_i = \{1, ..., m_i\}$, $i \in K$), and $V_i = \{v_{i,1}, ..., v_{i,m_i}\}$ is the set of all potential instances of VNF $i$. For convenience, $B_i = \{v_i\}$ is the unit set of the LB positioned before VNF $i$. Hence, $V^e = \bigcup_{i \in K} B_i \cup V_i$.

*Virtual edges of $G^e$.* We classify the edges of $G^e$ into four groups. The first refers to edges between LBs and their successor VNF instances, and is expressed as $E_{BV} = \bigcup_{i \in K} \bigcup_{j \in K_i} (v_i, v_{i,j})$. Further, we consider the edges between VNF instances and the LB placed behind the next VNF, denoted by $E_{VB} = \bigcup_{i \in K-\{k\}} \bigcup_{j \in K_i} (v_{i,j}, v_{i+1})$. Additionally, we take into account the event in which two adjacent VNFs share the same number of instances; in this case, the insertion of an LB between the VNFs is omitted, and connectivity can be established with a one-to-one fashion. To this end, we define $E_{1:1} = \bigcup_{i \in K-\{k\}} \bigcup_{j \in min(K_i, K_{i+1})} (v_{i,j}, v_{i+1,j})$ to be the set of edges that explicitly connect the respective instances of adjacent VNFs, where:

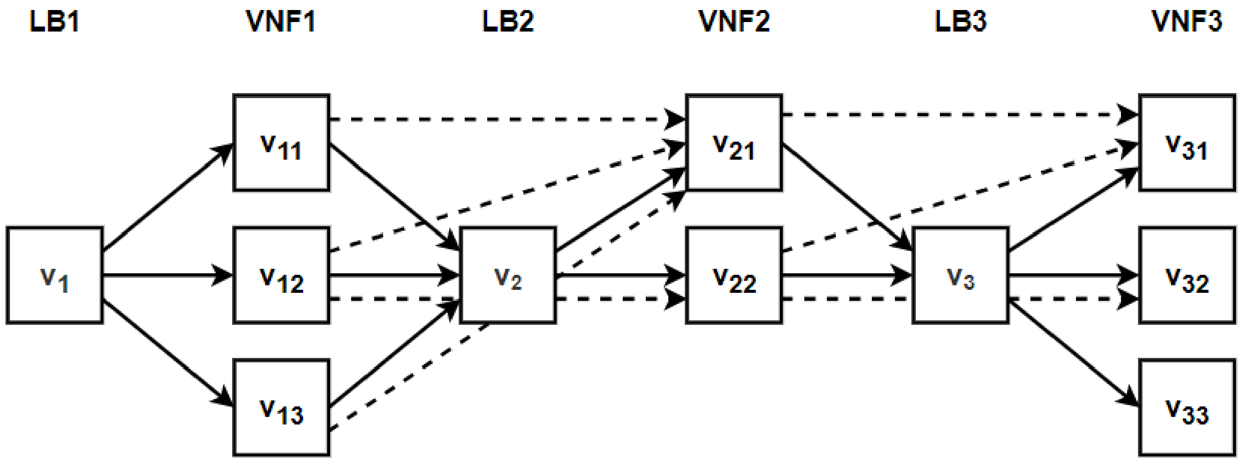
$$min(K_i, K_{i+1}) = \{K_i,\ if\ |K_i| \leq |K_{i+1}|,\ else\ K_{i+1}\}$$

Last, if a VNF $i$ ($i \in K - \{1\}$) comprises a single instance, there is no need to insert an LB before it. As such, the instances of the $i-1^{th}$ VNF can be directly connected to this VNF (*i.e.,* with a many-to-one fashion). Such edges are expressed with the set $E_{n:1} = \bigcup_{i \in K-\{k\}} \bigcup_{j \in K_i} (v_{i,j}, v_{i+1,1})$. Consequently, the set of edges in $G^e$ is $E^e = E_{BV} \cup E_{VB} \cup E_{1:1} \cup E_{n:1}$.

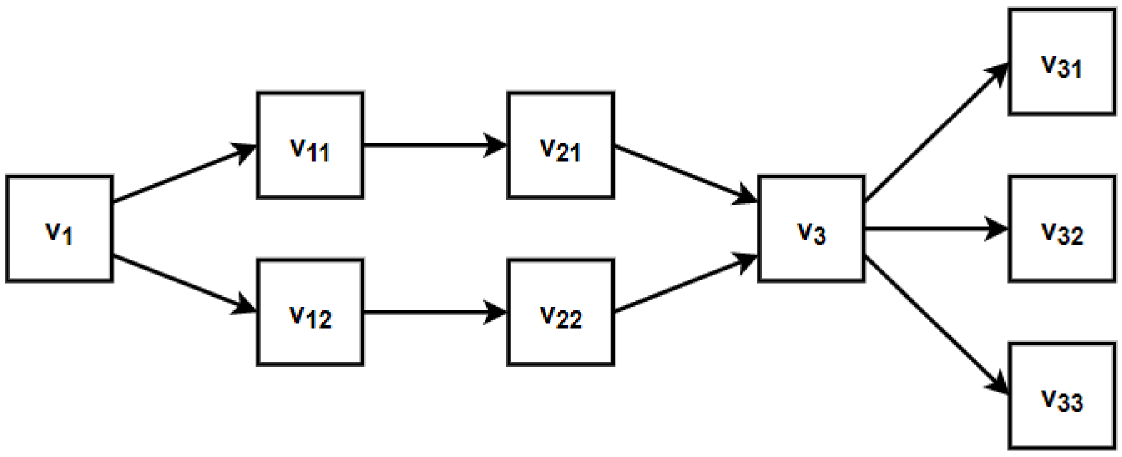
An illustration of our E-SFC model is given in Fig. 6.3. The initial SFC graph $G$ (Fig. 6.3a) consists of three VNFs. Each VNF is associated with an $m$-value, which indicates the maximum number of instances it can encompass. Here, the first and the third VNF can span at most three instances, whereas *VNF2* can be decomposed in up to two instances. The E-SFC graph $G^e$, generated based on $G$, is shown in Fig. 6.3b. LBs (*i.e.,* $v_1$, $v_2$, and $v_3$) are interposed between consecutive VNFs in the E-SFC graphs in order to facilitate potential connectivity among varying number of VNF instances.



(a) Initial SFC $G$



(b) Extended SFC (E-SFC) $G^e$



(c) A feasible transformation of $G$, $G'$

Figure 6.3: The initial SFC, the extended SFC, and a feasible transformation.

### 6.4.2 Feasible transformations

The essential purpose of the E-SFC is to serve as an auxiliary graph, upon which a selection of vertices and edges is performed. The actual selection of such elements shall ensure that the transformed graph and the original graph (i) exercise the same flow processing policy $\pi_f$ (*i.e.,* the VNF order is preserved), and (ii) have an (almost) equal performance $p$ (*i.e.,* the allocated resources to the decomposed elements sum up to the initial). The following definitions formalize these requirements:

**Definition 6.1.** *($\epsilon$-equivalent SFC graphs).* Two SFC graphs $G_1$ and $G_2$ are $\epsilon$-equivalent if they adhere to the exact same flow processing policy, *i.e.,* $\pi_{f_1} = \pi_{f_2}$, and they have a similar processing capacity, *i.e.,* $|p_1 - p_2| \leq \epsilon$, for some small positive $\epsilon$. We write $G_1 \sim G_2$ to denote that $G_1$ is $\epsilon$-equivalent to $G_2$.

We should note that, the interposition of LBs between VNFs does not affect the flow processing policy of an SFC, since LBs are neither meant to drop nor to modify packets. Consequently:

**Definition 6.2.** *(Feasible transformation).* Given an SFC $G$ and its associated E-SFC $G^e$, a feasible transformation of $G$ is every graph $G'$, such that $G' \subset G^e$ and $G' \sim G$. The set of all feasible transformations of $G$ is denoted as $\mathbb{T}(G)$.

The latter definition essentially imposes that feasible transformations shall be searched in $G^e$. For instance, the graph in Fig. 6.3c is a feasible transformations of $G$ in Fig. 6.3a, as long as the resource demands of the decomposed instances add up to the initial requirements of the respective VNFs. In particular, *VNF1* is decomposed into two instances, namely $v_{11}$ and $v_{12}$. Therefore, an LB $v_1$ needs to be inserted prior to these instances in order to steer the traffic appropriately. *VNF2* is also implemented with two instances ($v_{21}$ and $v_{22}$). The insertion of LB $v_2$ is not required, since the instances of *VNF1* and *VNF2* can be connected in a one-to-one fashion. However, this is not the case for *VNF2* and *VNF3*, given that the latter is decomposed into three instances, hence the LB $v_3$ needs to be interposed between the respective instances.

Notice that this is only one indicative feasible transformation of $G$, while the entire feasible transformations set consists of eighteen graph alternatives. In principle, $|\mathbb{T}(G)| = \prod_{i=1}^{k} m_i$

holds. The core challenge here is to identify criteria for assessing feasible transformations and, subsequently, to design algorithms that obtain optimized on par with the state of the underlying infrastructures.

### 6.4.3 Embedding flexibility

As discussed in Section 6.4.3, more expanded versions of SFCs yield higher flexibility in terms of embedding, since there are more options for embedding smaller VNF instances onto a substrate network. As such, an SFC-GT method should opt for the transformation of the initial SFC graph in a way that greatly facilitates its embedding.

However, such a transformation is by no means straightforward. More precisely, the transformation needs to take into account not only the properties of the SFC graph, but also the condition (*i.e.,* utilization, fragmentation level) of the physical infrastructure. Therefore, we introduce the function $F : (G', G, G_s) \mapsto \mathbb{R}$, which takes as input a feasible transformation of $G$ (*i.e.,* $G'$), the initial graph $G$, and the substrate network $G_s$, and is defined as follows:

$$F(G', G, G_s) = \phi(G_s) \cdot \frac{|V'| - |V|}{\sum_{i \in K} m_i - |V|} \tag{6.1}$$

where $\phi(G_s) \in [0, 1]$ denotes the percentage of servers in $G_s$ with scarce resources, while $|V'|$ encompasses the total number of nodes (excluding LBs) that form $G'$. The fraction in Eq. (6.1) applies a min-max normalization on the value of $|V'|$.

For the trivial case of an entirely unutilized datacenter, the embedding flexibility of each $G' \in \mathbb{T}(G)$ will be the same, since $\phi(G_s)$ will be zero. For non-trivial cases (*i.e.,* with a certain amount of CPU utilization), $\phi(G_s) \neq 0$. As $\phi(G_s)$ increases, more weight will be given on the embedding flexibility of $G'$, which is partially captured by $|V'|$. Note that $\phi(G_s)$ can be also adjusted to represent fragmentation [96], the average gap size [97], or other resource utilization indicators.

### 6.4.4 Transformation complexity

In principle, we consider several complexity aspects that can be attributed to a particular SFC graph structure. For instance, opting for $G$ over $G'$ (Figs. 6.3a and 6.3c) is expected to simplify service deployment and run-time service management, because of the smaller number of VNF instances that need to be spawned and managed. Furthermore, the number of instances affects the amount of state that an NFV orchestrator needs to maintain. In this work, we focus on what we deem as the most critical complexity aspect of SFC-GT, namely the resource overhead. In the following, we discuss the most significant factors that generate resource overhead. In this respect, assume that $G$ represents the initial SFC graph, whereas $G' \in \mathbb{T}(G)$.

**Load balancing.** According to the E-SFC model, we insert virtualized LBs before VNFs to split the incoming traffic among the instances of each VNF. To prevent packet re-ordering, we rely on flow-based load balancing, which can be realized in a stateless manner (*e.g.,* similar to ECMP). With such an LB scheme, a flow aggregate can be easily split among a set of VNF instances with the same processing capacity. One possible implication of flow-based LB is that traffic may not be distributed evenly among the VNF instances, in the case of significant diversity in the flow sizes. This problem can be rectified by detecting large flows and directing them to separate instances [98]. Such LB mechanisms require additional functionality and state, which, in turn, would generate extra resource overhead.

Since LB per se is not the focus of this paper, we incorporate a stateless flow-based LB mechanism in our model. Since the main computationally-intensive task within such an LB is packet I/O, we have generated a resource profile driven from packet I/O computational requirements (*i.e.,* 1300 CPU cycles/packet, according to [99, 100]). The resource overhead introduced by the insertion of LBs in the E-SFC is computed as follows:

$$B(G', G) = \frac{|R'| - |R|}{|R_{max}| - |R|} \tag{6.2}$$

In Eq. (6.2), $|R'|$ and $|R|$ express the total CPU demands of $G'$ and $G$, respectively, whereas $|R_{max}|$ denotes the CPU requirements of a feasible transformation of G that utilizes each and every LB from $G^e$ (*i.e.*, such a graph would result in the maximum LB overhead).

**TCAM consumption.** VNF chaining requires the installation of flow entries in switch TCAMs [3, 101, 102]. A larger number of VNF instances is expected to lead to increased TCAM consumption. VNF instance co-location could reduce TCAM consumption; however, this is subject to capacity constraints which become more severe as the utilization of the underlying infrastructure increases. We note that in commodity switches TCAM is usually limited (*i.e.,* few thousands of flow entries). The depletion of TCAM space can be potentially mitigated by the deployment of datapaths on commodity servers [103, 104]. However, this would lead to consumption of server capacity, which could have been utilized for VNF deployment instead.

In the context of SFC-GT, feasible transformations of a given graph $G$ will typically encompass at least as many edges as $G$. With the largest embedding footprint (typical worst-case embedding scenario), each pair of adjacent nodes will be placed in servers within different racks and, as a consequence, each edge will be mapped onto a path that connects these two servers. In a two-layer datacenter fat-tree topology, such a path will span three switches (two top-of-the-rack and one core), hence, three flow entries will be required for each edge of the graph $G$. Observing the extended graph (Fig. 6.3b) derived from a simplistic initial graph, we expect a significant overhead in terms of TCAM consumption, in the case of a highly expanded graph. We denote this overhead as $M(G', G)$ and compute it:

$$M(G', G) = \frac{|E'| - |E|}{|E_{max}| - |E|} \tag{6.3}$$

where $|E'|$ and $|E|$ express the number of edges of $G'$ and $G$, respectively, while $|E_{max}|$ corresponds to the maximum number of edges that can exist in a feasible graph of $G$.

**VNF state.** The majority of VNFs (*e.g.,* firewall, IDS, NAT) require internal state for their packet processing operations [105, 106]. Spawning additional instances of a stateful VNF requires, at the simplest case, the replication of their states across all instances, which leads to additional server memory consumption. Main memory consumption can be alleviated by correlating flows with running instances, such that each instance can maintain only a subset of the total state [107]. However, this yields higher complexity, since it requires the joint optimization of VNF instance placement and flow distribution. In our work, we account for the overhead stemming from state replication. The respective resource overhead is estimated as follows:

$$S(G', G) = \frac{s(G') - s(G)}{s(\underset{G_i \in \mathbb{T}(G)}{argmax} |V^i_{stateful}|) - s(G)} \tag{6.4}$$

where $s(\cdot)$ expresses the number of stateful VNF instances that belong in an arbitrary SFC. Additionally, we utilize $|V^i_{stateful}|$ to denote the number of stateful instances in graph $G_i \in \mathbb{T}(G)$. Therefore, $s(\underset{G_i \in \mathbb{T}(G)}{argmax} |V^i_{stateful}|)$ is the maximum number of stateful instances that can exist in a feasible transformation of $G$.

Essentially, Eq. (6.2), (6.3) and (6.4) apply a min-max normalization on the first terms of their nominators; thus, they share the same co-domain, which is $[0, 1]$. Since the resource overhead is considered to stem from load balancing, TCAM consumption, and VNF state replication, it is computed as:

$$C(G', G) = \alpha \cdot B(G', G) + \beta \cdot M(G', G) + \gamma \cdot S(G', G) \tag{6.5}$$

where $\alpha, \beta, \gamma \in \mathbb{R}_{\geq 0}$, $\alpha + \beta + \gamma = 1$, and $C : \ (G', G) \mapsto [0, 1]$.

## 6.5 SFC-GT optimization

In this section, we model the SFC-GT optimization problem as a mixed-integer linear program (MILP). The MILP shall ensure that the resulting transformation is indeed a feasible one, and the major challenge here lies in modelling a multitude of logical constraints in the form of linear expressions.

Table 6.1: Notations in the SFC, the E-SFC, and the MILP.

| **Symbol** | **Description** |
|---|---|
| *SFC* | |
| $V$ | the set of virtual nodes comprising a SFC |
| $E$ | the set of virtual edges between virtual nodes |
| $d(i)$ | CPU demand of virtual node $i$ |
| $d^i$ | inbound traffic to VNF $i$ |
| *E-SFC* | |
| $m_i$ | maximum number of instances allowed for VNF $i$ |
| $K$ | index set of VNFs, *i.e.*, of $V$ |
| $K_i$ | index set of VNF $i$ instances |
| $v_i$ | the $i^{th}$ LB node |
| $v_{i,j}$ | the $j^{th}$ instance of VNF $i$ |
| $B_i$ | the singleton comprising $v_i$ |
| $V_i$ | the set of $v_{i,j}$ instances |
| $V^e$ | the set of virtual nodes comprising the extended graph |
| $E_{BV}$ | the set of edges associated with $v_i$ and $v_{i,j}$ nodes |
| $E_{VB}$ | the set of edges associated with $v_{i,j}$ and $v_{i+1}$ nodes |
| $E_{VV}$ | the set of edges associated with $v_{i,j}$ and $v_{i+1,j}$ nodes |
| $E_{V1}$ | the set of edges associated with $v_{i,j}$ and $v_{i+1,1}$ nodes |
| $E^e$ | the set of virtual edges comprising the extended graph |
| *MILP* | |
| $x_i$ | assignment of node $v_i$ |
| $x_{i,j}$ | assignment of node $v_{i,j}$ |
| $r_{i,j}$ | CPU demand assigned to the $j^{th}$ instance of VNF $i$ |
| $z^i_{i,j}$ | assignment of edge $(v_i, v_{i,j}) \in E_{BV}$ |
| $r^i_{i,j}$ | resource demand assigned to edge $(v_i, v_{i,j})$ |
| $z^{i,j}_{i+1}$ | assignment of edge $(v_{i,j}, v_{i+1}) \in E_{VB}$ |
| $r^{i,j}_{i+1}$ | resource demand assigned to edge $(v_{i,j}, v_{i+1})$ |
| $z^{i,j}_{i+1,j}$ | assignment of edge $(v_{i,j}, v_{i+1,j}) \in E_{VV}$ |
| $r^{i,j}_{i+1,j}$ | resource demand assigned to edge $(v_{i,j}, v_{i+1,j})$ |
| $z^{i,j}_{i+1,1}$ | assignment of edge $(v_{i,j}, v_{i+1,1}) \in E_{V1}$ |
| $r^{i,j}_{i+1,1}$ | resource demand assigned to edge $(v_{i,j}, v_{i+1,1})$ |

### 6.5.1 Core variables

We initially introduce the core variables of our MILP program, which are practically associated with the assignment of nodes and edges from $G^e$, as well as the assignment of the respective resources. We denote by $x$ the binary variables that indicate the assignment of nodes, and by $z$ the binary variables that indicate the assignment of edges between nodes. In particular, $x_i$ refers to the assignment of $v_i$, *i.e.*, the $i^{th}$ LB, whereas $x_{i,j}$ denotes the assignment of $v_{i,j}$, *i.e.*, the $j^{th}$ instance of VNF $i$. Similarly, $z^i_{i,j}$ indicates the assignment of the edge that connects $v_i$ and $v_{i,j}$, $z^{i,j}_{i+1}$ expresses the assignment of the edge that connects $v_{i,j}$ and $v_{i+1}$, and, last, $z^{i,j}_{i+1,j}$

and $z^{i,j}_{i+1,1}$ refer to edges in $E_{VV}$ and $E_{V1}$, respectively. Each of the aforementioned variables (except from the variables $x_i$) is associated with an $r$-variable that expresses the amount of resources assigned to the corresponding virtual element. For example, $r_{0,2}$ refers to the CPU resources assigned to $v_{0,2}$, whereas $r^{0,2}_1$ represents the bandwidth requirements assigned to the link $(v_{0,2}, v_1)$.

### 6.5.2 Node and edge assignment constraints

**Reduction of the solution space.** We assume that instances of a particular VNF have the same processing capacity. Thus, according to the E-SFC model, there are many solutions that essentially express the same graph. For instance, the graph $v_{1,1} \to v_{2,1} \to v_{3,1}$ is the same as $v_{1,3} \to v_{2,1} \to v_{3,1}$, *i.e.,* both graphs encompass the same number of instances for each VNF (cf. Fig. 6.3c). Constraint (6.6) is defined to ensure that VNF instances are selected in ascending order. This reduces the solution space and alleviates the solver from evaluating solutions with the same efficiency.

$$x_{i,j+1} \leq x_{i,j} \quad \forall j \in K_i - \{m_i\}, \forall i \in K \tag{6.6}$$

**At least one instance of each VNF is selected.** Considering Eq. (6.6), this constraint is defined as shown in Eq. (6.7).

$$\sum_{i \in K} x_{i,1} = k \tag{6.7}$$

**LBs are not placed before VNFs with a singe instance.** This constraint, which eliminates unnecessary nodes, is defined as:

$$x_i + \epsilon \leq \sum_{j \in K_i} x_{i,j} \quad \forall i \in K, \epsilon \in (0,1) \tag{6.8}$$

**Assigned nodes enforce the selection of links, and vice-versa (Case: LB$_i$ $\rightarrow$ VNF$_i$).** Constraint (6.9) ensures that a virtual link is established between $v_i$ and the selected $v_{ij}$:

$$x_i \cdot x_{i,j} = z^i_{i,j} \quad \forall (v_i, v_{i,j}) \in E_{BV} \tag{6.9}$$

Since Eq. (6.9) is a non-linear constraint and all of the associated variables are binary, Eq. (6.9) can be linearized with the following constraints:

$$z^i_{i,j} \leq x_i \quad \forall (v_i, v_{i,j}) \in E_{BV} \tag{6.10}$$

$$z^i_{i,j} \leq x_{i,j} \quad \forall (v_i, v_{i,j}) \in E_{BV} \tag{6.11}$$

$$x_i + x_{i,j} - 1 \leq z^i_{i,j} \quad \forall (v_i, v_{i,j}) \in E_{BV} \tag{6.12}$$

**Assigned nodes enforce the selection of links, and vice-versa (Case: VNF$_i$ $\rightarrow$ LB$_{i+1}$).** In case a LB is placed before VNF $i+1$ (*i.e.*, $x_{i+1} = 1$), the selected instances $v_{i,j}$ of VNF $i$ should be connected to it. This is defined as:

$$x_{i+1} \cdot x_{i,j} = z^{i,j}_{i+1} \quad \forall (v_{i,j}, v_{i+1}) \in E_{VB} \tag{6.13}$$

We employ the previous logic, *i.e.*, similar to constraints (6.10) - (6.12), in order to linearize Eq. (6.13).

**LBs are not placed between adjacent VNFs with the same number of instances.**

This constraint enables the creation of parallel paths within a SFC graph, thus, allowing for more flexible (in terms of embedding) and robust graph structures. This is formulated via the following expression:

$$\sum_{j \in K_i} x_{i,j} = \sum_{j \in K_{i+1}} x_{i+1,j} \Rightarrow x_{i+1} = 0 \quad \forall i \in K - \{k\} \tag{6.14}$$

which certainly does not conform to LP modelling requirements. To this end, we introduce the integer variables $\delta^i_{i+1}$, $\delta^{i+}_{i+1}$, $\delta^{i-}_{i+1}$, and the binary variables $y^i_{i+1}$, as follows:

$$\delta^i_{i+1} = \sum_{j \in K_i} x_{i,j} - \sum_{j \in K_{i+1}} x_{i+1,j} \quad \forall i \in K - \{k\} \tag{6.15}$$

$$\delta^i_{i+1} = \delta^{i+}_{i+1} - \delta^{i-}_{i+1} \quad \forall i \in K - \{k\} \tag{6.16}$$

$$0 \leq \delta^{i+}_{i+1} \leq max(m_i, m_{i+1}) \cdot y^i_{i+1} \quad \forall i \in K - \{k\} \tag{6.17}$$

$$0 \leq \delta^{i-}_{i+1} \leq max(m_i, m_{i+1}) \cdot (1 - y^i_{i+1}) \quad \forall i \in K - \{k\} \tag{6.18}$$

$$x_{i+1} \leq \delta^{i+}_{i+1} + \delta^{i-}_{i+1} \quad \forall i \in K - \{k\} \tag{6.19}$$

Essentially, constraints (6.15) to (6.18) impose that $\delta^+ + \delta^- = |\delta| = |\sum_{j \in K_i} x_{i,j} - \sum_{j \in K_{i+1}} x_{i+1,j}|$. Consequently, Eq. (6.19) implies Eq. (6.14).

**Assigned nodes enforce the selection of links, and vice-versa (Case: VNF$_i$ $\rightarrow$ VNF$_{i+1}$).** It is further required to enforce the selection of edges that directly connect the respective instances of two adjacent VNFs, in case these are split into the same number of instances. Recall that the latter is implied by $\delta^{i+}_{i+1} + \delta^{i-}_{i+1} = 0$ (see constraint (6.14)). To this end, we introduce the following expression in our formulation:

$$(x_{i,j} = 1 \land x_{i+1,j} = 1) \land \delta_{i+1}^{i+} + \delta_{i+1}^{i-} = 0$$
$$\iff z_{i+1,j}^{i,j} = 1 \quad \forall j \in K_i, \forall i \in K - \{k\} \tag{6.20}$$

which can be easily linearized by the insertion of $t_{i+1,j}^{i,j}$ binary variables to hold the product of $x_{i,j}$ and $x_{i+1,j}$ (in a similar manner to the linearization of Eq. (6.9) and Eq (6.13)). In particular, the following expressions linearize constraint (6.20):

$$t_{i+1,j}^{i,j} - \delta_{i+1}^{i+} - \delta_{i+1}^{i-} \leq max(m_i, m_{i+1}) \cdot z_{i+1,j}^{i,j}$$
$$\forall (v_{i,j}, v_{i+1,j}) \in E_{VV} \tag{6.21}$$
$$\delta_{i+1}^{i+} + \delta_{i+1}^{i-} - t_{i+1,j}^{i,j} + \epsilon \leq max(m_i, m_{i+1}) \cdot (1 - z_{i+1,j}^{i,j})$$
$$\forall (v_{i,j}, v_{i+1,j}) \in E_{VV}, \epsilon \in (0,1) \tag{6.22}$$

**Assigned nodes enforce the selection of links, and vice-versa (Case: VNF$_i$ $\rightarrow$ VNF$_{i+1,1}$).** The following constraint enforces the required binding between the $x$ and $z$ variables, when there is only a single instance of VNF $i+1$, *i.e.*, $x_{i+1,2} = 0$ (see constraint (6.6)). In this case, according to constraint (6.8), we do not place a LB before the single instance. As such, the assigned $v_{i,j}$ instances, *i.e.*, those for which $x_{i,j} = 1$, should be directly connected to the $v_{i+1,1}$ node. This is expressed as follows:

$$x_{i+1,2} = 0 \land x_{i,j} = 1 \iff z_{i+1,1}^{i,j} = 1$$
$$\forall j \in K_i, \forall i \in K - \{k\} \tag{6.23}$$

which is equivalent to the following linear expressions:

$$x_{i,j} - x_{i+1,2} \leq z^{i,j}_{i+1,1} \quad \forall (v_{i,j}, v_{i+1,1}) \in E_{V1} \tag{6.24}$$

$$z^{i,j}_{i+1,1} \leq x_{i,j} \quad \forall (v_{i,j}, v_{i+1,1}) \in E_{V1} \tag{6.25}$$

$$z^{i,j}_{i+1,1} \leq 1 - x_{i+1,2} \quad \forall (v_{i,j}, v_{i+1,1}) \in E_{V1} \tag{6.26}$$

**Place an LB before the first VNF with multiple instances.** The first VNF in a SFC is treated as a special case, since the placement of a LB is required before the VNF, in case it encompasses multiple instances, *i.e.*, $\sum_{j\in K_1} x_{1,j} \geq 2$. In LP terms, this is formulated as:

$$\sum_{j\in K_1} x_{1,j} - 2 + \epsilon \leq m_1 \cdot x_1 \quad \epsilon \in (0,1) \tag{6.27}$$

**Place an LB between two VNFs with different number of instances and when the second VNF has multiple instances.** According to the previous discussions, we formulate this constraint as:

$$\begin{gathered} \delta^{i+}_{i+1} + \delta^{i-}_{i+1} \geq 1 \wedge x_{i+1,2} = 1 \Rightarrow x_{i+1} = 1 \\ \forall i \in K - \{k\} \end{gathered} \tag{6.28}$$

which is equivalent to the linear constraints:

$$x_{i+1} \leq x_{i+1,2} \cdot max(m_i, m_{i+1}) \quad \forall i \in K - \{k\} \tag{6.29}$$

$$\begin{aligned} &\delta^{i+}_{i+1} + \delta^{i-}_{i+1} - (1 - x_{i+1,2}) \cdot max(m_i, m_{i+1}) \\ &\qquad \leq x_{i+1} \cdot max(m_i, m_{i+1}) \quad \forall i \in K - \{k\} \end{aligned} \tag{6.30}$$

along with constraint (6.19).

### 6.5.3 Resource assignment constraints

**Resource demands for VNF instances.** Apparently, the resource demands of all VNF instances must sum up to the initial resource demands of the respective VNF. Recall that we opt for VNF instances with the same processing capacity, hence, their resource demands will be equal.

$$\sum_{j \in K_i} r_{i,j} \cdot x_{i,j} = d(i) \quad \forall i \in K \tag{6.31}$$

$$x_{i,j+1} = 1 \Rightarrow r_{i,j} = r_{i,j+1}$$
$$\forall j \in K_i - \{m_i\}, \forall i \in K \tag{6.32}$$

Yet, we should pay attention to the product of $r$ and $x$ variables in order to avoid non-linearity. To this end, we introduce $h_{i,j}$, which is a positive continuous variable adhering to:

$$h_{i,j} \leq x_{i,j} \cdot d(i) \quad \forall i \in K \tag{6.33}$$

$$r_{i,j} - (1 - x_{i,j}) \cdot d(i) \leq h_{i,j} \quad \forall i \in K \tag{6.34}$$

$$h_{i,j} \leq r_{i,j} \quad \forall i \in K \tag{6.35}$$

Constraints (6.33) - (6.35) ensure that $h_{i,j}$ holds the product $r_{i,j} \cdot x_{i,j}$. Therefore, Eq. (6.31) can be now expressed as:

$$\sum_{j \in K_i} h_{i,j} = d(i) \quad \forall i \in K \tag{6.36}$$

while Eq. (6.32) can be enforced by the following expressions in conjunction with constraint (6.6):

$$r_{i,j+1} \le r_{i,j} \quad \forall j \in K_i - \{m_i\}, \forall i \in K \tag{6.37}$$

$$r_{i,j} - r_{i,j+1} \le d(i) \cdot (1 - x_{i,j+1}) \quad \forall j \in K_i - \{m_i\}, \forall i \in K \tag{6.38}$$

$$r_{i,j+1} \le d(i) \cdot x_{i,j+1} \quad \forall j \in K_i - \{m_i\}, \forall i \in K \tag{6.39}$$

**Resource demands for edges departing from VNFs.** The bandwidth demands of edges that depart from VNF $i$ instances must sum up to $d^{i+1}$, *i.e.*, the total ingress traffic to VNF $i+1$. To enforce this, we introduce the following constraints:

$$\sum_{j \in K_i} (z^{i,j}_{i+1} \cdot r^{i,j}_{i+1} + z^{i,j}_{i+1,1} \cdot r^{i,j}_{i+1,1}) + \sum_{j \in min(K_i, K_{i+1})} z^{i,j}_{i+1,j} \cdot r^{i,j}_{i+1,j} = d^{i+1} \quad \forall i \in K - \{k\} \tag{6.40}$$

The individual products of $z$ and $r$ variables can be linearized in the same manner with Eqs. (6.33) - (6.35) for Eq. (6.31). However, we omit these nine constraints (three per product), for brevity. Since we employ constraint (6.32), we distribute the required bandwidth evenly among all edges, *i.e.*, edges that depart from VNF $i$ have equal resource demands. This is captured by the following expressions:

$$z_{i+1}^{i,j+1} = 1 \Rightarrow r_{i+1}^{i,j} = r_{i+1}^{i,j+1}$$
$$\forall j \in K_i - \{m_i\}, \forall i \in K - \{k\} \tag{6.41}$$
$$z_{i+1,1}^{i,j+1} = 1 \Rightarrow r_{i+1,1}^{i,j} = r_{i+1,1}^{i,j+1}$$
$$\forall j \in K_i - \{m_i\}, \forall i \in K - \{k\} \tag{6.42}$$
$$z_{i+1,j+1}^{i,j+1} = 1 \Rightarrow r_{i+1,j}^{i,j} = r_{i+1,j+1}^{i,j+1}$$
$$\forall j \in min(K_i, K_{i+1}) - \{min(m_i, m_{i+1})\}, \forall i \in K - \{k\} \tag{6.43}$$

Each one of the Eqs. (6.41), (6.42), and (6.43) can be applied in our MILP program through triplets of constraints (omitted for brevity), similar to Eqs. (6.37), (6.38), and (6.39).

**Resource demands for edges departing from LBs.** We further consider the bandwidth demands of edges that depart from the assigned LBs. In this respect, the following expression implies flow conservation, *i.e.,* the amount of inbound traffic to the LB is equal to the total amount of outbound traffic. This is formulated as:

$$\sum_{j \in K_i} z_{i,j}^{i} \cdot r_{i,j}^{i} = d^i \cdot z_{i,1}^{i}, \quad \forall i \in K \tag{6.44}$$

We linearize Eq. (6.44) by incorporating three additional constraints (similar to Eqs. (6.33) - (6.35)) in order to model the product between $z_{i,j}^i$ and $r_{i,j}^i$, whereas another triplet of constraints, in a similar manner to Eqs. (6.37) - (6.39), is utilized to enforce equal bandwidth demands among the edges that depart from LBs.

## 6.5.4 Objective function

The objective of the SFC transformation problem is as follows. Given an SFC $G$, a substrate network $G_s$, and its resource utilization state $\phi(G_s)$, identify $G^* \in \mathbb{T}(G)$, such that

$f(G^*, G, G_s) \leq f(G', G, G_s), \forall G' \in \mathbb{T}(G)$, where

$$f(G', G, G_s) = (1 - \phi(G_s)) \cdot C(G', G) - F(G', G, G_s) \tag{6.45}$$

Recall that, as explained in Section 6.4, $C$ and $F$ are contradictory; thus, our MILP seeks *Pareto-optimal* solutions (specifically, it tries to minimize the complexity, while maximizing the embedding flexibility of the transformation). Furthermore, since $F(G', G, G_s)$ depends heavily on $\phi(G_s)$, the value of Eq. (6.45) will be mainly driven by $C(G', G)$ whenever $\phi(G_s)$ is close to zero. However, as $\phi(G_s)$ deviates from zero, $F(G', G, G_s)$ will have a greater impact on the fitness score of candidate solutions.

## 6.6 Evaluation Results

In this section, we aim at quantifying the gains from our proposed SFC transformation method. To this end, we utilize an SFCE method in the following two evaluation scenarios: (i) our SFC-GT scheme generates an optimized transformation of the initial SFC, which is subsequently conveyed to the SFCE method (termed as *GT-augmented* embedding), and (ii) the initial SFC is processed directly by the SFCE method, without any SFC transformation (termed as *baseline* embedding).

The SFCE method is essentially a heuristic that computes SFC embeddings using the following steps. The heuristic maps the VNFs of the SFC in a sequential manner, starting with the first VNF of the SFC. Initially, it ranks the racks of the datacenter in descending order, according to their available ToR to core link capacity, and also ranks the servers of the current rack in descending order, according to their residual CPU capacity. Subsequently, the heuristic places the first VNF in the top ranked server, and prioritizes the assignment of adjacent VNFs in the same server. If such assignments are infeasible, the heuristic seeks another server of the current rack; otherwise, it repeats the process at the ranked servers of the subsequent rack, until each

VNF has been successfully placed. In case the placement of any VNF (or link) in the SFC fails, the embedding request is rejected.

### 6.6.1 Evaluation Environment

We have developed a simulation environment for SFC embedding evaluations in Python. The SFC-GT MILP is implemented with the Gurobi [58] solver. All evaluations are carried out on a simulated two-layer fat-tree datacenter network topology. The datacenter comprises 200 servers, organized in ten racks. The corresponding top-of-the-rack (ToR) switches are interconnected through five core switches. The intra-rack and inter-rack links have capacity of 1 Gbps and 10 Gbps, respectively, and the CPU capacity of each server is set to 7.2 GHz.

Each SFC consists of three to eight VNFs, whereas the CPU demand of each VNF in the SFC lies between 10 to 50% (with 10% step) of 7.2 GHz. The inbound traffic in the SFC lies anywhere between 50 and 200 Mbps. Concerning the maximum number of instances that each VNF can span (*i.e.,* $m_i$-values), we set these to five, while there is 80% probability that a VNF is stateful (otherwise, it is stateless). Furthermore, $\phi(G_s)$ indicates the percentage of servers with available CPU between 3% and 15%, whereas for the complexity metric, we set $\alpha, \beta$, and $\gamma$ to 0.4, 0.4, and 0.2, respectively.

In addition, we account for a set of $n$ discrete time intervals $T = \{1, 2, ..., n\}$. At each time interval, $N$ SFCs arrive sequentially, where $N$ follows a Poisson distribution with $\lambda = 20$. Each SFC comes with a lifespan $k$. Therefore, if an SFC is embedded at time $t \in T$, it will expire at time $t + k \in T$. During our simulations, $k$ lies within [3,10] (randomly), whereas $n$ is set to 500.

### 6.6.2 Evaluation Results

Initially, we compare the request acceptance rate achieved by the GT-augmented method and the baseline. This metric expresses the ratio of the number of successfully embedded SFCs

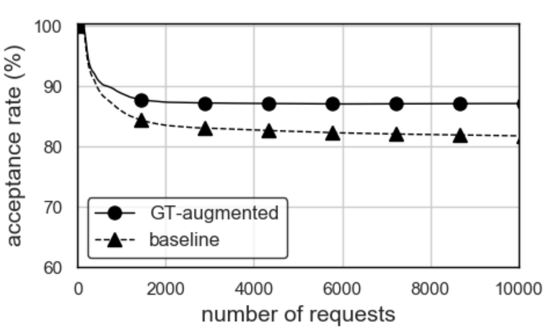


Figure 6.4: Request acceptance rate.

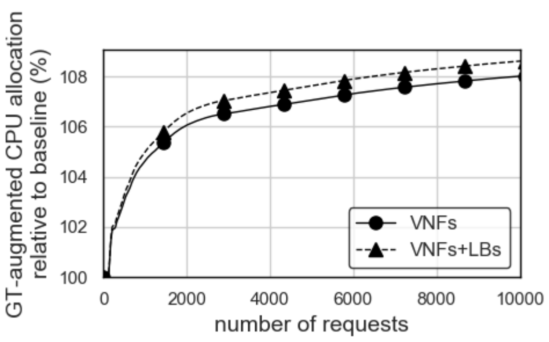


Figure 6.5: CPU utilization of GT-augmented relative to baseline.

over the total number of SFCs. According to Fig. 6.4, GT-augmented admits 87% of the requests, whereas the respective percentage for the baseline is 82%. Further examination of the acceptance rate results uncovers that all rejections of GT-augmented are attributed to lack of CPU resources, whereas a small fraction of the baseline rejections is caused by lack of bandwidth, as well.

As expected, the higher acceptance rate of the proposed method is translated into higher resource efficiency, as shown in Fig. 6.5. In this plot, we compute the moving average of the fractions $CPU_i^1/CPU_i^2$, where $CPU_i^1$ is the sum of the CPU assigned to SFCs, until request $i$ by the GT-augmented method (the denominator captures the same value for the baseline). Recall that a graph transformation may introduce further CPU requirements, because of the addition of LBs. The reasonable assumption is that such resources are not monetized by the InPs, who may opt for utilizing them in favor of higher resource efficiency. That said, Fig. 6.5 indicates that the GT-augmented algorithm manages to assign 9% more CPU compared to the baseline, while the net gain (*i.e.,* resources for VNFs, which are indeed monetized) is 8%.

Fig. 6.6 illustrates the relative efficiency of the proposed method against the baseline in terms of generated traffic, sub-divided into intra- and inter-rack. In this plot, the respective values are computed per time interval (which is per ≈20 SFCs), thereby, covering the whole range of the simulation. While initially both curves tend to fluctuate, they eventually reach a steady state, which is the same for both DC topology levels, *i.e.,* at 90% of the traffic generated by the baseline. Apparently, this indicates higher efficiency for the proposed method, especially

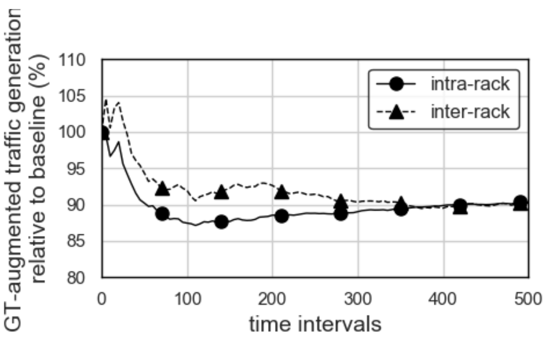


Figure 6.6: Generated traffic by GT-augmented relative to baseline.

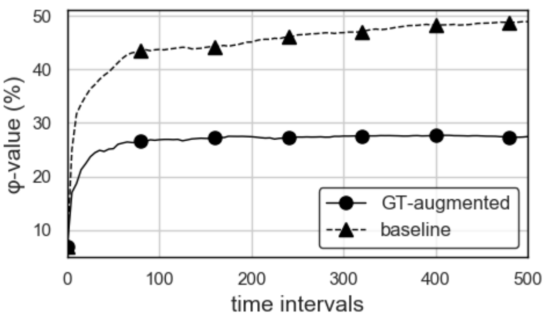


Figure 6.7: $\phi$-values of GT-augmented and baseline.

at the inter-rack level, where bandwidth conservation is crucial. At first, bandwidth conservation through VNF decomposition (into multiple instances) may not be expected, especially since more emphasis was given (narrative-wise) on relaxing the CPU requirements for enhanced embedding flexibility. Nevertheless, SFC-GT may indeed generate less traffic, since VNF decomposition opens up opportunities for co-locating more communicating SFC components into the same server or rack. For instance, each one of the links $v_{11} \rightarrow v_{21}$ and $v_{12} \rightarrow v_{22}$ in Fig. 6.3c yields half of the bandwidth requirements compared to the link $VNF1 \rightarrow VNF2$ in Fig. 6.3a. Therefore, if $VNF1$ and $VNF2$ are not placed on the same server, the whole required bandwidth will be allocated from the corresponding DC links. In contrast, assume that the GT-augmented method places only $v_{12}$ and $v_{22}$ on different servers. This would result in only half of the bandwidth consumption observed in the previous case (*i.e.,* without employing GT-augmented).

To gain further insights on the behavior of the two methods, we examine the condition of the underlying DC, quantified by the $\phi$-values, *i.e.,* the percentage of servers with available CPU between 3% and 15%, which we use as a rough approximation for CPU fragmentation. According to Fig. 6.7, $\phi$ reaches 50% with the baseline (with a tendency to increase), whereas the respective value with the GT-augmented is significantly lower. Additional results (which are omitted for brevity) indicate that, at steady state, the baseline utilizes 80% of the servers' CPU (the respective value with GT-augmented is 90%). This, along with the measured high fragmentation, implies that a considerable portion of the residual CPU is scattered in resource

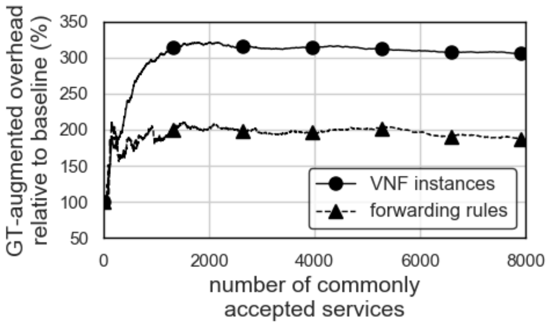


Figure 6.8: VNF instances and forwarding rules of GT-augmented relative to baseline.

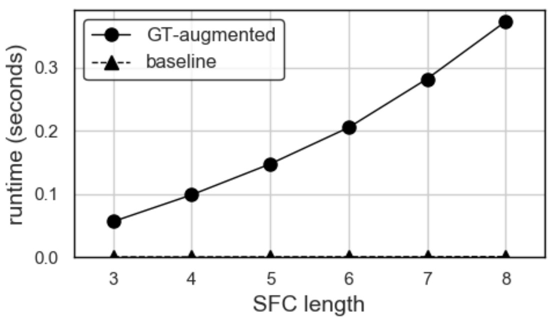


Figure 6.9: Average solver run-time for diverse SFC lengths.

blocks of size 3% to 15%, a fact that highly restricts the embedding options of SFCs, eventually reducing the efficiency of the baseline.

Recall that SFC-GT seeks a balance between maximizing the embedding flexibility and minimizing the transformation complexity of graph transformations. So far, our evaluation results corroborate the gains of SFC-GT in terms of embedding flexibility. With respect to transformation complexity, Fig. 6.8 illustrates the generated VNF instances, as well as the required forwarding rules for the proposed method, in relation to the initial service graph and its mapping by the baseline. For this comparison, the successful mapping of the respective graphs (*i.e.,* transformed and initial) is crucial, hence, we only account for the commonly accepted SFCs by both methods. In this respect, we observe 3x increase in the number of VNF instances, and 2x increase in terms of forwarding rules. Practically, this means that the transformed graphs embedded by GT-augmented comprise of triple the nodes of the initial graph, while their chaining requires twice as much forwarding rules on the switches, on average. Given that $m_i$-values are set to 5, this result indicates an achieved balance in terms of VNF instances. In terms of forwarding rules, steering traffic across more VNF instances inevitably increases switch TCAM consumption. In future work, we will seek to alleviate this TCAM consumption overhead, by coupling SFC-GT with source routing in an experimental environment.

Finally, we compare the solver run-time between the GT-augmented and the baseline. According to Fig. 6.9, the run-time of GT-augmented exhibits an exponential growth, in relation to the problem size (*i.e.,* SFC length). However, for reasonable SFC lengths and values of $m_i$, the GT-augmented run-time is bounded below 0.4 sec, which indicates its feasibility in realistic embedding scenarios.

## 6.7 Related Work

To the best of our knowledge, this is the first work that tackles SFC-GT through the lens of embedding flexibility and transformation complexity. Therefore, this section discusses research which focuses on the general problem class that SFC-GT falls under, namely NFV-RA preprocessing problems (cf. Section 2.3.4).

Mehraghdam *et al.* [108] propose an SFC specification and placement framework. In terms of SFC specification, which is more relevant to our work, the authors identify the possibility of optionally ordered VNFs, which can be deployed in parallel modules. To this end, they aim at splitting a VNF into multiple instances, while positioning a load balancer node beforehand. The final SFC graph conveyed to their placement algorithm is obtained with a heuristic method that orders VNFs in a way that greedily strives to minimize the bandwidth requirements of the SFC.

Carpio *et al.* [94] examine the potential of creating replicas of sequential SFCs and placing them into distinct network locations, thus balancing the traffic across multiple SFC instances. Sun *et al.* [93] propose a VNF parallelism framework, which dedicates a core module for the identification of VNF dependencies and eventually generates an SFC that allows the concurrent execution of VNFs. The authors provide a comprehensive analysis of their solution, while the perks of parallelism are corroborated through an evaluation of their prototype. None of these studies, though, takes into consideration the impact of the SFC modification on the embedding procedure.

Palkar *et al.* [95] present E2, a framework that facilitates the deployment of SFCs. Among the E2 features, VNF placement includes a sizing step that determines the number of instances required for each VNF. Yet, this number is computed as a function of the projected VNF load, thus no relevant optimization occurs during a pre-embedding phase.

Despite focusing on distinct problems, a common factor of the aforementioned studies is the prospective gains from the transformation of the initial SFC request. In contrast to these studies which investigate specific aspects of SFC transformation, we provide a holistic solution for optimized SFC transformation, accounting for a wide range of embedding flexibility and transformation complexity aspects, with the ultimate goal to strike a balance between them.

## 6.8 Conclusions

Effective resource assignment is crucial for NFV. Resource allocation challenges, such as resource fragmentation and high utilization, highlight the shortcomings of traditional methods. By transforming SFC graphs before embedding, we offer an innovative method to mitigate these issues. The primary goal is to maximize resource allocation efficiency without introducing unnecessary overheads.

At the heart of our approach to SFC-GT are two key principles: embedding flexibility and transformation complexity. The former measures how easily an SFC graph can be embedded into a physical network, and the latter captures the additional overheads of the transformed graph. A significant portion of our research revolves around balancing these factors, influenced by the current resource state of the network. This balance ensures efficient utilization of physical resources without introducing unnecessary overhead in the network. From a practical standpoint, our SFC-GT method assists SFCE solvers to find feasible solutions which adhere to the embedding objective in cases where network resources are highly utilized. In the absence of our SFC-GT method, SFCE solvers would be at a significant disadvantage, having limited means to address these situations.

Our methodology covers foundational concepts up to rigorous mathematical modelling. Using an MILP model provides a structured approach, laying the groundwork for deeper insights. Our simulation results validate the efficacy of the SFC-GT approach. These not only highlight potential resource utilization improvements but also quantify any associated overheads, offering a rounded perspective on the method's practical applications. This research offers promising strategies for refining resource assignment in the evolving world of NFV.

# Chapter 7

# Conclusions

## 7.1 Summary of Thesis Achievements

In the evolving landscape of modern networking, the task of embedding SFCs within NFV-enabled infrastructures has garnered significant attention. This is especially pertinent in the realm of resource-constrained environments like edge clouds, which introduce unique challenges due to their inherent limitations in computational capabilities, storage, and bandwidth. In such scenarios, efficient SFCE becomes imperative to ensure service delivery with high performance, while utilizing available resources optimally. This thesis delves deep into the nuances of SFCE in these constrained environments, addressing a set of critical challenges and proposing innovative solutions to them. The specific challenges, along with the way these are addressed via the research of the thesis, are listed below:

1. A prevailing assumption in core datacenters is the abundance of certain resource types apart from the CPU, such as memory and storage. This perspective has driven many existing SFCE studies to focus primarily on optimizing a singular resource type, namely the CPU. These studies assert that their solvers can be seamlessly expanded to handle multi-resource environments. However, in the face of resource-constrained clouds, the validity of this assumption is put to the test, calling for a more comprehensive evaluation.

To address this challenge, the thesis examines a range of multi-dimensional mapping efficiency metrics, evaluating their integration in both heuristic and exact SFCE solvers. Harnessing the most appropriate and efficient metrics, we introduce two heuristic methods and an MILP tailored for multi-dimensional SFCE. To further refine VNF placement, a VNF bundling scheme is proposed to leverage the perks of two critical mapping efficiency metrics, namely the euclidean distance and the cosine similarity. Empirical evaluations showcase a considerable improvement in resource efficiency with the multi-dimensional heuristics in contrast to single-dimensional strategies, albeit with a slight sub-optimality when compared against the MILP. An exploration of the bundling scheme's synergy with our most prominent heuristic further solidifies the value and potential implications of integrating multi-dimensional metrics in SFCE methodologies.

2. Edge cloud environments present an opportunity for cross-service interactions, which can boost network capabilities. However, edge nodes, like servers, often have limited resources. This makes it essential to allocate resources carefully, especially when many SFCs interact with one another. Added to this is the dynamic nature of edge clouds, which require flexible scaling mechanisms that can adjust to the needs of interacting services. This situation adds new challenges to NFV. There is a clear need to look at SFCE with this perspective to ensure efficient use of resources in the next-generation networks. To tackle this, we develop a CSC-aware SFCE heuristic that optimizes SFC placement. This new approach goes beyond looking at resources and communication demands within a single service, and it also considers the embedding constraints imposed by deployed SFCs that might be consumed. We elaborate on various types of CSC and introduce a new data structure called the VNF embedding tree. This structure helps decide the best order for placing the VNFs of SFCs, allowing the heuristic to manage the complexities of SFCs that have CSC needs. Simulation results indicate that the SFCE-CSC heuristic manages to optimize the placement of CSC constructs without reducing the embedding efficiency of the original SFCs.

3. RL has gained traction in the sphere of SFCE, with a predominant incline towards centralized approaches. A common assumption is that a centralized agent has a complete

view of the environment, which spans multiple interconnected edge clouds. However, this stands in contrast to the NFV MANO paradigm, wherein a centralized agent is more likely to access merely aggregate data, offering a partial view of the actual network state. Such an inconsistency highlights the urgency to re-examine the alignment of RL schemes with practical orchestration frameworks. Addressing this gap, our research initially sheds light on the motivations behind the popularity of RL in SFCE and subsequently identifies inherent drawbacks in prevailing RL-based SFCE strategies. We then argue that these limitations lie in the centralized execution of RL methodologies. In response, we introduce a cooperative DMARL model tailored for decentralized SFCE. This model emphasizes the pivotal role of efficient communication amongst neighboring agents. Evaluations via simulations underscore that DMARL surpasses the performance of a centralized double deep Q-learning mechanism. They also uncover the core behaviors acquired by the team of agents, emphasize the value of agent-to-agent information exchange, and draw attention to the effects of varying network topologies on the efficacy of DMARL.

4. The prevailing literature reveals a significant oversight concerning SFC pre-processing techniques. As it stands, SFC requests are largely treated as rigid entities, with their structure assumed to be static and unchanging. Such a viewpoint critically hinders the adaptability and versatility of SFCE solutions, preventing their optimization in harmony with the real-time resource availability within the physical network. The situation calls for innovative methodologies capable of reshaping SFC requests to better align with fluctuating network conditions, thereby enhancing the capacity of solvers to obtain optimal solutions, even under unfavourable network states. Addressing this gap, our work delves into the potential to transform entire SFC graphs prior to their embedding. This aims to assist SFCE solvers obtain more effective solutions, or to obtain solutions more efficiently. The overarching transformation process is majorly influenced by the resource consumption status of the substrate network, along with the concepts of embedding flexibility and transformation complexity. We model the SFC graph transformation problem as an MILP, and we detail the linearization process of multiple non-linear constraints. Rigorous simulations validate the viability of our method and highlight the resource efficiency

advancements that can be realized even at highly utilized and fragmented infrastructures.

## 7.2 Applications

The advanced mechanisms and methodologies discussed in this thesis hold significant potential, transcending theoretical constructs to offer tangible solutions to real-world challenges. From optimizing network functionalities to expanding the horizons of adaptable computing, the scope of these mechanisms is broad and far-reaching.

Within the realm of NFV, the potential of these mechanisms is particularly promising. Envision a scenario where a series of typical SFCs, whether confined to a single edge datacenter or distributed across multiple ones, need orchestration. Leveraging the NFV MANO framework, the SFCE algorithms of this research can be integrated seamlessly within resource schedulers of either VIMs (in case of a single datacenter) or NFVOs (in case of multiple datacenters). Here, they can play a pivotal role, adeptly handling not just the placement of these SFCs but also their efficient scaling (recall that SFCE solutions are paramount in horizontal scaling), ensuring resource optimization and reduced latency. With the rise of edge computing and 5G technologies, such scenarios are not just likely but imminent, and the methodologies developed herein can be instrumental in bridging the gap between demand and optimized service delivery.

Beyond the specific domain of SFC graphs, the mechanisms can be adapted to address more generic application graphs. Consider services and applications that might not traditionally fit the ecosystem of NFV but still require orchestrated, efficient embedding and management. From complex computational workflows in data science and bioinformatics, where tasks have dependencies and need to be executed in specific sequences, to sophisticated digital twins in industrial IoT setups that necessitate real-time data processing and analytics, the methodologies can be repurposed. They can offer optimized resource allocation, adaptability, and scalability, ensuring that regardless of the nature of the application, its demands are met with precision and efficiency.

## 7.3 Future Work

While the mechanisms presented in this thesis have demonstrated significant advancements in orchestrating SFCs in resource-constrained NFV environments, several intriguing paths for further exploration remain, including:

- *Granularity of Resource Allocation.* While our research primarily focuses on macro-level resource allocation in edge cloud environments, there is potential to delve into micro-level allocations. Indicative examples of such allocations include last level cache and memory bandwidth management. This will unveil the way individual resources within a server or container are allocated and optimized, especially in hyper-converged infrastructures.
- *Multi-layer Orchestration.* As evidenced in Chapter 5, the interplay of NFVOs and VIMs with regard to resource allocation is far from trivial, and additional research is required to understand and identify the sweet-spot between the granularity of information each layer shall have at its disposal and bandwidth conservation.
- *Inter-connected decision-making systems.* Inter-connected decision-making systems strive to bridge isolated units of intelligence, aiming for a cohesive and global optimization strategy. However, a paramount research challenge lies in quantifying the downstream effects of these systems, ensuring that local actions do not compromise overarching objectives. This necessitates the development of new metrics and methodologies capable of capturing the intricate consequences of inter-connectivity.

Apparently, the continuous evolution of technology and the ever-changing demands of modern network architectures ensure that the journey of discovery, innovation, and refinement is far from over.

# Bibliography

[1] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: Network processing as a cloud service," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 13–24, 2012.

[2] M. Chiossi *et al.*, "Network functions virtualisation: An introduction, benefits, enablers, challenges & call for action," *ETSI White Paper*, 2012.

[3] C. Papagianni, P. Papadimitriou, and J. S. Baras, "Rethinking service chain embedding for cellular network slicing," in *2018 IFIP Networking Conference (IFIP Networking) and Workshops*. IEEE, 2018, pp. 1–9.

[4] J. G. Herrera and J. F. Botero, "Resource allocation in nfv: A comprehensive survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, 2016.

[5] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE communications surveys & tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[6] "ETSI NFV," https://www.etsi.org/technologies/nfv/, accessed: 2023-01-30.

[7] "Open Source MANO," https://osm.etsi.org/, accessed: 2023-01-30.

[8] "ONAP," https://www.onap.org/, accessed: 2023-01-30.

[9] A. Pentelas, G. Papathanail, I. Fotoglou, and P. Papadimitriou, "Network service embedding with multiple resource dimensions," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium.* IEEE, 2020, pp. 1–9.

[10] ——, "Network service embedding across multiple resource dimensions," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 209–223, 2020.

[11] A. Pentelas and P. Papadimitriou, "Network service embedding for cross-service communication," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM).* IEEE, 2021, pp. 424–430.

[12] A. Pentelas, D. De Vleeschauwer, C.-Y. Chang, K. De Schepper, and P. Papadimitriou, "Deep multi-agent reinforcement learning with minimal cross-agent communication for sfc partitioning," *IEEE Access*, 2023.

[13] A. Pentelas and P. Papadimitriou, "Service function chain graph transformation for enhanced resource efficiency in nfv," in *2021 IFIP Networking Conference (IFIP Networking).* IEEE, 2021, pp. 1–9.

[14] G. Papathanail, I. Fotoglou, C. Demertzis, A. Pentelas, K. Sgouromitis, P. Papadimitriou, D. Spatharakis, I. Dimolitsas, D. Dechouniotis, and S. Papavassiliou, "Cosmos: An orchestration framework for smart computation offloading in edge clouds," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium.* IEEE, 2020, pp. 1–6.

[15] I. Fotoglou, G. Papathanail, A. Pentelas, P. Papadimitriou, V. Theodorou, D. Dechouniotis, and S. Papavassiliou, "Towards cross-slice communication for enhanced service delivery at the network edge," in *2020 6th IEEE Conference on Network Softwarization (NetSoft).* IEEE, 2020, pp. 22–28.

[16] G. Papathanail, A. Pentelas, I. Fotoglou, P. Papadimitriou, K. V. Katsaros, V. Theodorou, S. Soursos, D. Spatharakis, I. Dimolitsas, M. Avgeris *et al.*, "Meson: Optimized cross-slice communication for edge computing," *IEEE Communications Magazine*, vol. 58, no. 10, 2020.

[17] L. Beraldo, A. Pentelas, F. L. Verdi, P. Papadimitriou, and C. A. Marcondes, “Tenant-oriented resource optimization for cloud network slicing with performance guarantees,” in *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*. IEEE, 2021, pp. 38–44.

[18] M. Bitzi, M.-E. Xezonaki, V. Theodorou, I. Dimolitsas, D. Dechouniotis, S. Papavassiliou, G. Papathanail, I. Fotoglou, A. Pentelas, and P. Papadimitriou, “Meson: Optimized cross-slice communication for pervasive virtual cdn services,” in *2021 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*. IEEE, 2021, pp. 1–2.

[19] G. Papathanail, A. Pentelas, and P. Papadimitriou, “Towards fine-grained resource allocation in nfv infrastructures,” in *2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2021, pp. 1–6.

[20] D. Laskaratos, I. Dimolitsas, G. Papathanail, M.-E. Xezonaki, A. Pentelas, V. Theodorou, D. Dechouniotis, T. Bozios, P. Papadimitriou, and S. Papavassiliou, “Meson: A platform for optimized cross-slice communication on edge computing infrastructures,” *IEEE Access*, vol. 10, pp. 49 322–49 336, 2022.

[21] “NFVO API,” https://nfvwiki.etsi.org/index.php?title=API_specifications/, accessed: 2023-01-30.

[22] “Openstack,” https://www.openstack.org/, accessed: 2023-01-30.

[23] S. Gyawali, S. Xu, Y. Qian, and R. Q. Hu, “Challenges and solutions for cellular based v2x communications,” *IEEE Communications Surveys & Tutorials*, vol. 23, no. 1, pp. 222–255, 2020.

[24] M. H. C. Garcia, A. Molina-Galan, M. Boban, J. Gozalvez, B. Coll-Perales, T. Şahin, and A. Kousaridas, “A tutorial on 5g nr v2x communications,” *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1972–2026, 2021.

[25] P. Popovski, K. F. Trillingsgaard, O. Simeone, and G. Durisi, "5g wireless network slicing for embb, urllc, and mmtc: A communication-theoretic view," *Ieee Access*, vol. 6, pp. 55 765–55 779, 2018.

[26] P. Yang, X. Xi, K. Guo, T. Q. Quek, J. Chen, and X. Cao, "Proactive uav network slicing for urllc and mobile broadband service multiplexing," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 10, pp. 3225–3244, 2021.

[27] "Open Baton," https://openbaton.github.io/, accessed: 2023-01-30.

[28] M. A. T. Nejad, S. Parsaeefard, M. A. Maddah-Ali, T. Mahmoodi, and B. H. Khalaj, "vspace: Vnf simultaneous placement, admission control and embedding," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 542–557, 2018.

[29] M. Raeis, A. Tizghadam, and A. Leon-Garcia, "Reinforcement learning-based admission control in delay-sensitive service systems," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE, 2020, pp. 1–6.

[30] T. J. Wassing, D. De Vleeschauwer, and C. Papagianni, "A machine learning approach for service function chain embedding in cloud datacenter networks," in *2021 IEEE 10th International Conference on Cloud Networking (CloudNet)*. IEEE, 2021, pp. 26–32.

[31] D. Dietrich, A. Abujoda, A. Rizk, and P. Papadimitriou, "Multi-provider service chain embedding with nestor," *IEEE Transactions on Network and Service Management*, vol. 14, no. 1, pp. 91–105, 2017.

[32] Y. Sang, B. Ji, G. R. Gupta, X. Du, and L. Ye, "Provably efficient algorithms for joint placement and allocation of virtual network functions," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[33] R. Cziva, C. Anagnostopoulos, and D. P. Pezaros, "Dynamic, latency-optimal vnf placement at the network edge," in *Ieee infocom 2018-ieee conference on computer communications*. IEEE, 2018, pp. 693–701.

[34] B. Ren, D. Guo, Y. Shen, G. Tang, and X. Lin, “Embedding service function tree with minimum cost for nfv-enabled multicast,” *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 5, pp. 1085–1097, 2019.

[35] S. Agarwal, F. Malandrino, C.-F. Chiasserini, and S. De, “Joint vnf placement and cpu allocation in 5g,” in *IEEE INFOCOM 2018-IEEE conference on computer communications.* IEEE, 2018, pp. 1943–1951.

[36] L. Liu, S. Guo, G. Liu, and Y. Yang, “Joint dynamical vnf placement and sfc routing in nfv-enabled sdns,” *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4263–4276, 2021.

[37] “DPKD,” https://www.dpdk.org/, accessed: 2023-01-30.

[38] P. Soto, D. De Vleeschauwer, M. Camelo, Y. De Bock, K. De Schepper, C.-Y. Chang, P. Hellinckx, J. F. Botero, and S. Latré, “Towards autonomous vnf auto-scaling using deep reinforcement learning,” in *2021 Eighth International Conference on Software Defined Systems (SDS).* IEEE, 2021, pp. 01–08.

[39] X. Fei, F. Liu, H. Xu, and H. Jin, “Adaptive vnf scaling and flow routing with proactive demand prediction,” in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications.* IEEE, 2018, pp. 486–494.

[40] “Intel RDT,” https://www.intel.com/content/www/us/en/architecture-and-technology/resource-director-technology.html/, accessed: 2023-01-30.

[41] P. Sohal, M. Bechtel, R. Mancuso, H. Yun, and O. Krieger, “A closer look at intel resource director technology (rdt),” in *Proceedings of the 30th International Conference on Real-Time Networks and Systems*, 2022, pp. 127–139.

[42] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker, “E2: a framework for nfv applications,” in *Proceedings of the 25th Symposium on Operating Systems Principles*, 2015, pp. 121–136.

[43] A. F. Ocampo, J. Gil-Herrera, P. H. Isolani, M. C. Neves, J. F. Botero, S. Latré, L. Zambenedetti, M. P. Barcellos, and L. P. Gaspary, "Optimal service function chain composition in network functions virtualization," in *Security of Networks and Services in an All-Connected World: 11th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS 2017, Zurich, Switzerland, July 10-13, 2017, Proceedings 11.* Springer International Publishing, 2017, pp. 62–76.

[44] E. Coronado, R. Behravesh, T. Subramanya, A. Fernández-Fernández, S. Siddiqui, X. Costa-Pérez, and R. Riggio, "Zero touch management: A survey of network automation solutions for 5g and 6g networks," *IEEE Communications Surveys & Tutorials*, 2022.

[45] M. Rost and S. Schmid, "On the hardness and inapproximability of virtual network embeddings," *IEEE/ACM Transactions on Networking*, vol. 28, no. 2, pp. 791–803, 2020.

[46] D. Dietrich, C. Papagianni, P. Papadimitriou, and J. S. Baras, "Network function placement on virtualized cellular cores," in *2017 9th International Conference on Communication Systems and Networks (COMSNETS).* IEEE, 2017, pp. 259–266.

[47] D. Spatharakis, I. Dimolitsas, D. Dechouniotis, G. Papathanail, I. Fotoglou, P. Papadimitriou, and S. Papavassiliou, "A scalable edge computing architecture enabling smart offloading for location based services," *Pervasive and Mobile Computing*, vol. 67, p. 101217, 2020.

[48] R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder, "Heuristics for vector bin packing," *research. microsoft. com*, 2011.

[49] V. V. Vazirani, *Approximation algorithms.* Springer, 2001, vol. 1.

[50] S. Lee, R. Panigrahy, V. Prabhakaran, V. Ramasubramanian, K. Talwar, L. Uyeda, and U. Wieder, "Validating heuristics for virtual machines consolidation," *Microsoft Research, MSR-TR-2011-9*, pp. 1–14, 2011.

[51] F. Parreño, R. Alvarez-Valdés, J. F. Oliveira, and J. M. Tamarit, "A hybrid grasp/vnd algorithm for two-and three-dimensional bin packing," *Annals of Operations Research*, vol. 179, no. 1, pp. 203–220, 2010.

[52] M. H. Ferdaus, M. Murshed, R. N. Calheiros, and R. Buyya, "Virtual machine consolidation in cloud data centers using aco metaheuristic," in *European conference on parallel processing.* Springer, 2014, pp. 306–317.

[53] Z. Xiao, W. Song, and Q. Chen, "Dynamic resource allocation using virtual machines for cloud computing environment," *IEEE transactions on parallel and distributed systems*, vol. 24, no. 6, pp. 1107–1117, 2012.

[54] "Microsoft Azure," https://azure.microsoft.com/.

[55] "Amazon Web Services," https://aws.amazon.com/.

[56] "Google Cloud," https://cloud.google.com/.

[57] "IBM Cloud," https://www.ibm.com/cloud.

[58] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2019. [Online]. Available: http://www.gurobi.com

[59] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4. ACM, 2014, pp. 455–466.

[60] T. Benson, A. Akella, A. Shaikh, and S. Sahu, "Cloudnaas: a cloud networking platform for enterprise applications," in *Proceedings of the 2nd ACM Symposium on Cloud Computing.* ACM, 2011, p. 8.

[61] A. Gember, A. Akella, A. Anand, T. Benson, and R. Grandl, "Stratos: Virtual middleboxes as first-class entities," 2012.

[62] A. Abujoda and P. Papadimitriou, "Distnse: Distributed network service embedding across multiple providers," in *2016 8th International Conference on Communication Systems and Networks (COMSNETS).* IEEE, 2016, pp. 1–8.

[63] X. Fu, F. R. Yu, J. Wang, Q. Qi, and J. Liao, "Service function chain embedding for nfv-enabled iot based on deep reinforcement learning," *IEEE Communications Magazine*, vol. 57, no. 11, pp. 102–108, 2019.

[64] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, 2012.

[65] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types." in *Nsdi*, vol. 11, no. 2011, 2011, pp. 24–24.

[66] M.-A. Kourtis, M. J. McGrath, G. Gardikis, G. Xilouris, V. Riccobene, P. Papadimitriou, E. Trouva, F. Liberati, M. Trubian, J. Batallé *et al.*, "T-nova: An open-source mano stack for nfv infrastructures," *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 586–602, 2017.

[67] [Online]. Available: https://github.com/Peniac/NSE-CSC

[68] B. Agarwal, A. Akella, A. Anand, A. Balachandran, P. Chitnis, C. Muthukrishnan, R. Ramjee, and G. Varghese, "Endre: An end-system redundancy elimination service for enterprises." in *NSDI*, 2010, pp. 419–432.

[69] T. Truong-Huu, P. M. Mohan, and M. Gurusamy, "Service chain embedding for diversified 5g slices with virtual network function sharing," *IEEE Communications Letters*, vol. 23, no. 5, pp. 826–829, 2019.

[70] F. Malandrino, C. F. Chiasserini, G. Einziger, and G. Scalosub, "Reducing service deployment cost through vnf sharing," *IEEE/ACM Transactions on Networking*, vol. 27, no. 6, pp. 2363–2376, 2019.

[71] C. Benzaid and T. Taleb, "Ai-driven zero touch network and service management in 5g and beyond: Challenges and research directions," *IEEE Network*, vol. 34, no. 2, pp. 186–194, 2020.

[72] E. Coronado, R. Behravesh, T. Subramanya, A. Fernández-Fernández, S. Siddiqui, X. Costa-Pérez, and R. Riggio, "Zero touch management: A survey of network automation solutions for 5g and 6g networks," *IEEE Communications Surveys & Tutorials*, 2022.

[73] J. Pei, P. Hong, M. Pan, J. Liu, and J. Zhou, "Optimal vnf placement via deep reinforcement learning in sdn/nfv-enabled networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 263–278, 2019.

[74] L. Wang, W. Mao, J. Zhao, and Y. Xu, "Ddqp: A double deep q-learning approach to online fault-tolerant sfc placement," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 118–132, 2021.

[75] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proceedings of the tenth international conference on machine learning*, 1993, pp. 330–337.

[76] L. Matignon, G. J. Laurent, and N. Le Fort-Piat, "Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems," *The Knowledge Engineering Review*, vol. 27, no. 1, pp. 1–31, 2012.

[77] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls *et al.*, "Value-decomposition networks for cooperative multi-agent learning," *arXiv preprint arXiv:1706.05296*, 2017.

[78] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson, "Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning," in *International Conference on Machine Learning*, 2018, pp. 4295–4304.

[79] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[80] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, 2016.

[81] N. Toumi, M. Bagaa, and A. Ksentini, "Hierarchical multi-agent deep reinforcement learning for sfc placement on multiple domains," in *2021 IEEE 46th Conference on Local Computer Networks (LCN)*, 2021, pp. 299–304.

[82] P. T. A. Quang, Y. Hadjadj-Aoul, and A. Outtagarts, “A deep reinforcement learning approach for vnf forwarding graph embedding,” *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1318–1331, 2019.

[83] Z. Yan, J. Ge, Y. Wu, L. Li, and T. Li, “Automatic virtual network embedding: A deep reinforcement learning approach with graph convolutional networks,” *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1040–1057, 2020.

[84] J. Foerster, N. Nardelli, G. Farquhar, T. Afouras, P. H. Torr, P. Kohli, and S. Whiteson, “Stabilising experience replay for deep multi-agent reinforcement learning,” in *International conference on machine learning*, 2017, pp. 1146–1155.

[85] J. Zheng, C. Tian, H. Dai, Q. Ma, W. Zhang, G. Chen, and G. Zhang, “Optimizing nfv chain deployment in software-defined cellular core,” *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 248–262, 2019.

[86] J. Jia, L. Yang, and J. Cao, “Reliability-aware dynamic service chain scheduling in 5g networks based on reinforcement learning,” in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, 2021, pp. 1–10.

[87] S. Haeri and L. Trajković, “Virtual network embedding via monte carlo tree search,” *IEEE transactions on cybernetics*, vol. 48, no. 2, pp. 510–521, 2017.

[88] Y. Xiao, Q. Zhang, F. Liu, J. Wang, M. Zhao, Z. Zhang, and J. Zhang, “Nfvdeep: Adaptive online service function chain deployment with deep reinforcement learning,” in *Proceedings of the International Symposium on Quality of Service*, 2019, pp. 1–10.

[89] R. Solozabal, J. Ceberio, A. Sanchoyerto, L. Zabala, B. Blanco, and F. Liberal, “Virtual network function placement optimization with deep reinforcement learning,” *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 292–303, 2019.

[90] J. C. Cisneros, S. Yangui, S. E. P. Hernández, and K. Drira, “A survey on distributed nfv multi-domain orchestration from an algorithmic functional perspective,” *IEEE Communications Magazine*, vol. 60, no. 8, pp. 60–65, 2022.

[91] P. T. A. Quang, A. Bradai, K. D. Singh, and Y. Hadjadj-Aoul, “Multi-domain non-cooperative vnf-fg embedding: A deep reinforcement learning approach,” in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS).* IEEE, 2019, pp. 886–891.

[92] Y. Zhu, H. Yao, T. Mai, W. He, N. Zhang, and M. Guizani, “Multiagent reinforcement-learning-aided service function chain deployment for internet of things,” *IEEE Internet of Things Journal*, vol. 9, no. 17, pp. 15 674–15 684, 2022.

[93] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu, “Nfp: Enabling network function parallelism in nfv,” in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 43–56.

[94] F. Carpio, S. Dhahri, and A. Jukan, “Vnf placement with replication for loac balancing in nfv networks,” in *2017 IEEE international conference on communications (ICC).* IEEE, 2017, pp. 1–6.

[95] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker, “E2: A framework for nfv applications,” in *Proceedings of the 25th Symposium on Operating Systems Principles*, 2015, pp. 121–136.

[96] J. Gehr and J. Schneider, “Measuring fragmentation of two-dimensional resources applied to advance reservation grid scheduling,” in *IEEE/ACM CCGrid*, 2009.

[97] L. Tom, B. Caminero, C. Carrión *et al.*, “Improving grid resource usage: Metrics for measuring fragmentation,” in *IEEE/ACM CCGrid*, 2012.

[98] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, A. Vahdat *et al.*, “Hedera: dynamic flow scheduling for data center networks.” in *Nsdi*, vol. 10, no. 8. San Jose, USA, 2010, pp. 89–92.

[99] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy, “Routebricks: exploiting parallelism to scale software routers,” in *ACM SOSP*, 2009.

[100] A. Abujoda and P. Papadimitriou, “Profiling packet processing workloads on commodity servers,” in *IFIP WWIC*, 2013.

[101] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, “Simple-fying middlebox policy enforcement using sdn,” in *ACM SIGCOMM computer communication review*, vol. 43, no. 4, 2013, pp. 27–38.

[102] S. K. Fayazbakhsh, V. Sekar, M. Yu, and J. C. Mogul, “Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions,” in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013, pp. 19–24.

[103] N. Sarrar, S. Uhlig, A. Feldmann, R. Sherwood, and X. Huang, “Leveraging zipf’s law for traffic offloading,” *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 1, p. 16–22, Jan. 2012.

[104] Z. Bozakov and P. Papadimitriou, “Openvroute: An open architecture for high-performance programmable virtual routers,” in *IEEE HPSR*, 2013.

[105] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, “Opennf: Enabling innovation in network function control,” in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4. ACM, 2014, pp. 163–174.

[106] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield, “Split/merge: System support for elastic execution in virtual middleboxes,” in *USENIX NSDI*, Apr. 2013.

[107] Z. Cao, A. Abujoda, and P. Papadimitriou, “Distributed data deluge (d3): efficient state management for virtualized network functions,” in *IEEE INFOCOM SWFAN*, 2016.

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