Dynamic VNF Forwarding Graph Extension Algorithms

4.1 Introduction

Network Function Virtualization (NFV) coupled with Software-Defined Networking (SDN) is expected to impact the business to business to consumer relationships, current business models and value chains by enabling requests from multiple tenants for virtual infrastructures to deploy and provide their services to other providers and consumers.

Providers that deploy this variety of services and network functions in both physical and virtual infrastructures on behalf of tenants will have to treat via multiple network functions traffic originating, transiting and terminating in their infrastructures. The service provider has dynamic SFC establishment, update, and extension requirements and needs that the infrastructure providers have to meet. The latter will consequently need not only to remove, replace, insert, and add VNF instances, but also to extend already instantiated service chains without disturbing the previously deployed chains.

In this foreseen context, network providers (or physical infrastructure providers and more generally NFV architecture designers and actors) are actually faced with dynamic and variable demands of virtualized network infrastructures from tenants. These actors are indeed

likely to deploy their services, hosted in virtualized infrastructures, gradually as their business grows. These players not only need to control, classify, and steer user and application traffic flows in their dedicated slices but also want to extend their already acquired and operational slices with additional service graphs (adding new forwarding paths, new service chains, new services and virtualized network functions). These extensions of already hosted network function graphs have to be achieved without disrupting initially deployed and active service instances. The extensions must be seamless to applications and services that do not tolerate interruptions, migration or disruptions in quality of service (QoS) and experience.

In order to meet these requirements, this chapter proposes algorithms to extend already deployed network services and function graphs to respond to new demands while taking into account the constraint of minimizing the effect on the original service graphs.

According to [67] and [80], Virtual Network Function scaling and extension are triggered by new client requirements and the rising of network load over time. This is especially true for services with periodic resource demands (e.g., cloud applications such as an online shopping store during holiday seasons, etc.), and managing VNF workload changes [87] (e.g., during peak hours for cellular telecom operators).

In this chapter, we address the more generic and general problem of extending already deployed service function chains (or VNF-FGs) and aim at providing algorithms that can achieve various types of extensions. The latter may be a request to insert a new VNF in an existing graph, a new demand to extend a Network Forwarding Path (NFP) by adding VNFs to an existing chain, and more generally extend an already deployed VNF-FG with a complex service graph.

We refer to the graph extension as a VNF-FG extension to emphasize that we also address networking level extension of already deployed forwarding graphs in addition to the fact that we inherently answer the needs for service function chaining extensions. The extensions, as mentioned, can be triggered by new requests or by the VNF-FG life cycle management that may decide to extend the initial graph to adapt to network and traffic load changes such as adding load balancers, spawning new VNFs to absorb increasing load, etc.

We first propose an Integer Linear Programming (ILP) model as an exact formulation of the VNF-FG extension problem. The ILP can find optimal solutions for reasonable problem sizes and thus can serve as a reference for the performance evaluation and as a way to assess the quality of the solutions for heuristic algorithms at least for small graph sizes. The heuristics should be efficient, find good solutions, and scale with problem size much better than the ILP. To reduce complexity while finding good solutions, we resort to a Steiner Tree based algorithm and an Eigendecomposition algorithm using network adjacency matrices of the requested and hosting graphs as a basis for optimal embedding. The ILP and the heuristic algorithms ensure that the specified connecting points between the previously described graphs and the extensions are respected as specified in the client requests and that there are no disruptions (or minimal disruption) to the initially deployed VNF-FG. We compare the performance of these algorithms in terms of rejection rate of new requests, quality of the solutions and service disruptions as a function of infrastructure size, network connectivity, and system load.

Section 4.2 of this chapter presents the related work. The problem formulation is described in Section 4.3. The proposals are introduced in Section 4.4. Section 4.5 reports the performance evaluation results. Section 4.6 summarizes the main findings.

4.2 Related Work

Since we are addressing the extension of service function chains already deployed in hosting infrastructures, the related work review focusses on the virtual network topology change problem with emphasis on the dynamic expansion and adaptation of Virtualized Network Function-Forwarding Graphs (VNF-FGs) whenever relevant. Indeed, previous work addressed partially or simply did not address extension of already deployed services but rather initial placement, adaptation to changes of an existing graph and/or the hosting infrastructure and reaction to faults. We consequently limit the description to work as close as possible to ours while being aware that extensions of SFCs or VNF-FGs have not been directly and explicitly and fully addressed in the past.

In [88], authors present JASPER, a fully automated approach to jointly optimize scaling, placement, and routing decisions for complex network services. Two algorithms are developed for adaptation of existing services to changes in the demand, a Mixed Integer Linear Program (MILP) and a custom constructive heuristic.

Ayoubi et al. [89] introduced RELIEF, an availability-aware embedding and reconfiguration framework for elastic services in the cloud. The framework comprises two main modules. The JENA sub-system that performs virtual network (VN) embedding to provide just-enough availability guarantees based on the availability of the physical servers hosting the virtual one. The second module, ARES, is a reliable reconfiguration bloc that adapts the embedding of hosted services as they scale (by migrating the VN or adding backup nodes). This work does not address either, explicitly, graph extension requests from already served clients or increasing workloads which is our instead our central concern or objective.

Liu et al. [90] consider the readjustment of VNF chaining in dynamic situations and conditions by trying to optimize jointly the deployment of new SFCs and the readjustment of existing service chains in order to satisfy variable user requirements. This is closer to some extent to our work and stated goal. However, unlike our approach, they use pre-calculated paths in the network. We are focussing on the extension of already deployed chains and service graphs while keeping the initially provided and operating tenant graph unmodified, except for edges involved in the extension. The authors of [90] use first an ILP formulation to solve the problem exactly then reduce time complexity using a Column Generation (CG) heuristic algorithm that approximates the performance of the ILP. The main idea behind the CG based algorithm is to decompose the original problem into a master problem and a sub-problem to solve them iteratively in order to obtain a near-optimal solution.

In [91], authors present a comprehensive analysis of a resource allocation algorithm for VNFs based on genetic algorithms (GA) for the initial placement of VNFs and the scaling of VNFs to support traffic changes. For the scaling of existing policies proposed in [91], the algorithm starts with the current state and searches for the re-assignment of resources for the set of VNFs that need scaling. Therefore, some VNFs change their initial locations and the algorithm tries to minimize the number of server and link configuration changes (to minimize disruptions to existing traffic).

Li et al. [92] implemented a real-time resource provisioning system for NFV called NFV-RT, which integrates timing analysis with several techniques, such as service chain consolidation, and Integer Linear Programming with rounding. Also, the scaling is achieved by duplicating full instances of VNF chains, and by reshuffling chains and migrating traffic.

The proposals defined in this chapter differs from previous work and that of [91] and [92]. They actually address the problem of **Resource and Function Scaling** defined in [93] as the adaptation of the assigned network resources to functions or flows, or adaptation of the number of deployed instances of a specific VNF (by scale out, scale in, or migration). We are instead concerned by another type of flexibility, specifically **Topology Adaptation**, defined also in [93], by extending the graph structure of already active virtual networks that require an extension with additional nodes and links. Our addressed graph extension can contain new network and service functions and is far more complex than simply duplicating the service chain by a graph replication technique as proposed in [92]).

Note that all our algorithms, in this chapter, maintain the initial mapping location of already instantiated service chains (original VNF-FG) to avoid disturbing previously deployed and running services (i.e., without migrations or interruptions), to preserve previous deployments and ensure network stability.

In summary, our work differs significantly from the existing VNFs scaling and adaptation proposals since none really addresses the problem of extending an already deployed tenant VNF-FGs following a new request for seamless (non disrupting) evolutions of these dedicated service chains (i.e., maintaining acceptable service performance).

4.3 Problem Formulation

The problem of extending an already deployed and operating tenant SFC or VNF-FG corresponds to the placement of new requested VNFs and flow paths in the hosting infrastructure. This has to be accomplished while respecting all previous deployments and the specified connectivity between the initial graph and its extension. This is a placement problem with a set of very specific constraints. The placement of virtual functions and path extensions

must cover a subset of previously deployed graph nodes and links, those involved in interconnection of the old and new graph and the ingress and egress switches (or gateways) involved in both graphs. Hence, the objective is to embed a new graph in the infrastructure with a specific node cover requirement with respect to a previous graph deployment. The problem is intuitively identified as some form of node cover problem and a spanning tree problem. The solution must span or cover some specific nodes in the original SFCs and VNF-FGs and find an optimal mapping of the new request nodes and paths. To address this problem we use an ILP formulation that serves as a reference for performance and quality comparisons with two heuristic algorithms, a Steiner Tree solution and an Eigendecomposition approach. Both the Steiner and Eigendecomposition can take into account the old and new graphs inter-connectivity requirements. The eigendecomposition relies on adjacency matrices that reflect the networking topologies of the graphs and for this reason is a natural candidate to address VNF-FG extensions in shared infrastructures. The Steiner Tree approach adopted here corresponds to a generalization of the non-negative shortest path and the spanning tree problems and in fact to the well known Steiner tree problem in graphs.

4.3.1 Substrate Graph or NFV Infrastructure

In order to derive an ILP for the SFC extension problem, we start from a mathematical graph representation of the extension request, the previously deployed service graph (SFC or VNF-FG) and the hosting infrastructure. In addition to this, a description of the interconnection between the previously deployed graph and the extension graph is required, but we embed instead this in the constraints and conditions that must be respected by all algorithms when placing the extension in the rest of the hosting infrastructure.

The physical network (known as substrate and physical infrastructure and used interchangeably), as defined by the ETSI NFV Infrastructure (NFV-I) in [83], is modeled as an undirected weighted graph $G_p = (N_p, E_p)$ where E_p is the set of physical links and N_p is the set of physical nodes.

Each substrate node, $k \in N_p$, is characterized by its i) available processing capacity (i.e., CPU) CPU_k, and ii) type T_k: switch, server or Physical Network Function (PNF) [84]. The PNFs are the traditional physical middleboxes implementing network functions.

Each physical link (i.e., $e \in E_p$) is characterized by its available bandwidth BW_e.

An example of such an infrastructure, known as the NFV-I, including two PNFs and two switches (an ingress switch and an egress switch acting as input and output gateways for the VNF-FG or SFC related Network Forwarding Paths) and some interconnected servers is presented in Figure 4.1.

4.3.2 VNF Forwarding Graph

Similarly a client request (i.e., a requested Service Function Chain (SFC)) is modeled as a directed graph $G_v = (N_v, E_v)$ where N_v is the set of virtual nodes and E_v is the set of virtual links in the requested graph.

Each virtual node, $i \in N_v$, is characterized by its i) required processing power cpu_i and ii) its type t_i: VNF or switch (i.e., ingress or egress). Each virtual link $e_{ij} \in E_v$ is described by its required bandwidth $bw_{e_{ij}}$.

As specified by the SFC IETF working groups [94], we associate a VNF-type to each VNF to represent the network service or function type (e.g., firewall, DPI, NAT, load balancer, SSL, etc.) and their respective requirements. Note that the VNFs can be hosted only by servers or PNFs having the same type. The ingress and egress nodes can be hosted only by switches. Figure 4.1 depicts two Network Forwarding Paths (NFP). Each Forwarding Path NFP describes the ordered VNF sequence the traffic must pass through.

From the VNF forwarding graph G_v, we derive an intermediate request graph called the Network Connectivity Topology graph NCT_v. The $NCT_v = (N_v, E_v)$ is a weighted undirected graph having exactly the same set of nodes and edges than G_v. The key attribute of an NCT node $i \in N_v$ is its requested processing capacity cpu_i. The weight (or requested bandwidth $bw_{e_{ij}}$) of an NCT virtual link $e_{ij} \in E_v$ is the sum of the requested bandwidths of all the VNF flows passing through it. This representation means that there is no path

splitting over links connecting two consecutive VNFs and that the same physical link will be used to host the aggregate demand between any two VNFs for a given VNF-FG.

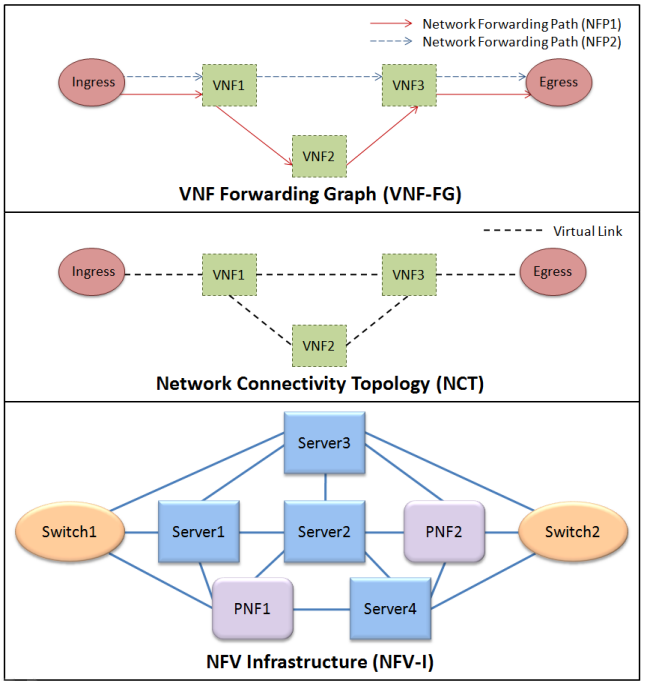


FIGURE 4.1: The VNF-FG and NFV-I topologies

In the ETSI-MANO document [95], the requested graphs are defined as follows:

- **Network connectivity topology (NCT)**: is a graph that specifies the VNF nodes that compose the global network service and the connection between these nodes through virtual links (VL). Each VL is connected to a VNF through a connection point (CP) that represents the VNF interface. Hence, an NCT defines a logical topology among VNFs in a network. Note that this logical topology represented by an NCT may change as a function of user requirements, business policies, and/or network context.

- **VNF Forwarding Graph (VNF-FG)**: is a graph established on top of the NCT. The VNF-FG is composed of network forwarding paths (NFP) that are ordered lists of connection points (CPs) forming a chain of VNFs.

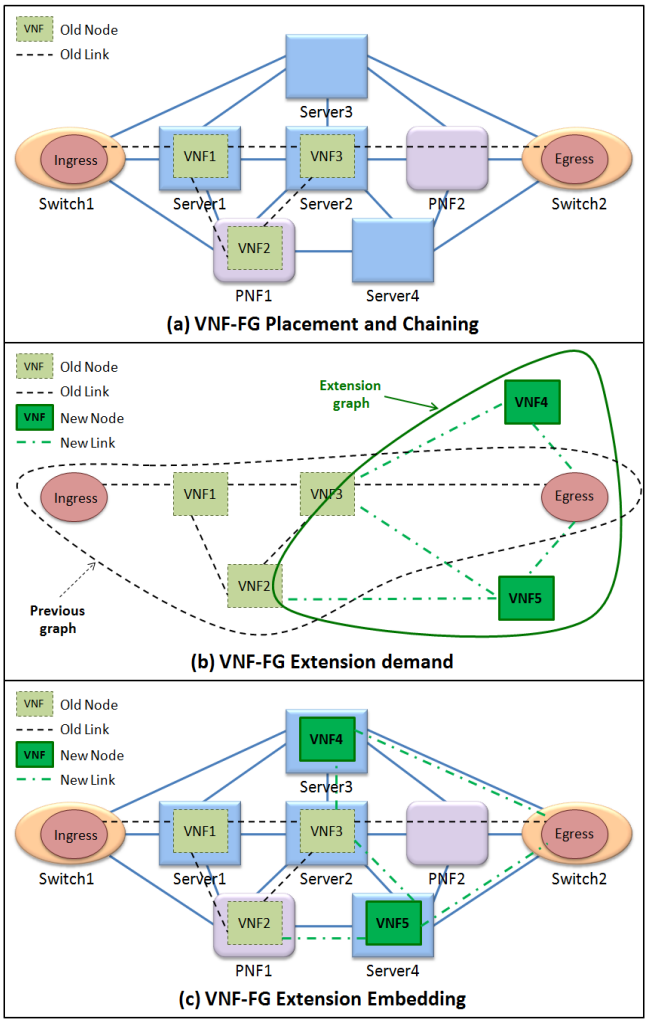


FIGURE 4.2: VNF Forwarding Graph Extension

4.3.3 VNF Forwarding Graph Extension

Figure 4.2 describes the type of VNF-FG extensions considered in our work. Figure 4.2(a) depicts the previously deployed forwarding graph (or slice) before receiving an extension request for two new VNFs (VNF 4 and VNF 5). Figure 4.2(b) shows their required connectivity with the already deployed graph. Figure 4.2(c) depicts a placement solution for the extension with VNF 4 and VNF 5 hosted respectively in Server 3 and Server 4. The newly requested interconnection links and forwarding paths are also highlighted.

4.4 Proposals

To solve the problem of VNF-FG extension placement and chaining, we propose an Integer Linear Programming model, a Steiner Tree based algorithm and an eigendecomposition approach that uses as a basis the adjacency matrices of the request graph and the hosting infrastructure graph.

4.4.1 ILP Model

We now formulate the ILP model for VNF-FG extension. Table 4.1 summarizes the parameters and the used variables. The model includes integrity constraints associated to the objective function used to achieve optimal extension.

Old node mapping constraint: This constraint maintains the initial mapping location of old VNF nodes.

$$x_{im_i} = 1, \quad \forall i \in N_v^{old} \tag{4.4.1}$$

Old link mapping constraint: maintains the initial mapping location of old virtual links.

$$y_{e_{ij},P_{m_i,m_j}} = 1, \quad \forall e_{ij} \in E_v^{old} \tag{4.4.2}$$

New node mapping constraint: ensures that new VNFs, VNF i, is mapped to exactly one physical candidate node k and that the VNF components (VNFCs composing the VNF) cannot be split into (distributed across) many physical nodes. This is expressed by:

$$\sum_{k \in \mathcal{C}} x_{ik} = 1, \quad \forall i \in N_v^{new} \tag{4.4.3}$$

where:

$$x_{ik} = \begin{cases} 1, & \text{if the VNF } i \text{ is hosted on the substrate node } k; \\ 0, & \text{otherwise.} \end{cases} \tag{4.4.4}$$

TABLE 4.1: Table of Notations

Notation	**Description**
CPU_k	Residual capacity in a physical node k
m_i, m_j	Initial mapping location of old VNFs i and j
P_{k_i,k_j}	Physical path interconnecting physical nodes k_i and k_j
$\mathcal{C}$	Set of physical nodes candidates
$\mathcal{P}$	Set of physical paths candidates
min_{CPU}	The minimum capacity in $\mathcal{C}$
BW_e	Residual bandwidth in one physical link e
δ_{ep}	A boolean parameter indicating if the physical link $e \in E_p$ belongs to path $P_{k_i,k_j} \in \mathcal{P}$: $\delta_{ep} = 1 \Leftrightarrow e \in P_{k_i,k_j}$
N_v^{new} E_v^{new}	Set of new virtual nodes Set of new virtual links
N_v^{old} E_v^{old}	Set of old virtual nodes Set of old virtual links
cpu_i	Required capacity by virtual node i
$bw_{e_{ij}}$	Required bandwidth between virtual nodes i and j
x_{ik}	A binary variable indicating whether VNF i is mapped to physical node k
$y_{e_{ij},P_{k_i,k_j}}$	A binary variable indicating whether virtual link e_{ij} is mapped to physical path P_{k_i,k_j}

Capacity constraint: makes sure that the residual capacity in a physical node k satisfies the required capacity by VNF i. This leads to the following inequality:

$$\sum_{i \in N_v^{new}} cpu_i \times x_{ik} \leq CPU_k, \quad \forall k \in \mathcal{C} \tag{4.4.5}$$

New link mapping constraint: *The case where both extremity of virtual link e_{ij} are new nodes:* Each new virtual link e_{ij} is mapped to exactly one physical path P_{k_i,k_j} where k_i is a physical candidate for the i^{th} VNF and k_j is a physical candidate for the j^{th} VNF. Note that

i and j are neighbors in the VNF-FG request.

$$\sum_{k_i \in \mathcal{C}} \sum_{k_j \in \mathcal{C}} y_{e_{ij}, P_{k_i,k_j}} = 1, \ \ \forall e_{ij} \in E_v^{new}, \ \forall i, j \in N_v^{new} \tag{4.4.6}$$

where:

$$y_{e_{ij}, P_{k_i,k_j}} = \begin{cases} 1, & \text{if the virtual link } e_{ij} \text{ is mapped to physical path } P_{k_i,k_j}; \\ 0, & \text{otherwise.} \end{cases} \tag{4.4.7}$$

The case where source extremity i of virtual link e_{ij} is an old node and destination extremity j is a new node: Each new virtual link e_{ij}, starting from old VNF i, is mapped to exactly one physical path P_{m_i,k_j} where m_i is the initial mapping of old VNF i and k_j is a physical candidate for the new VNF j.

$$\sum_{k_j \in \mathcal{C}} y_{e_{ij}, P_{m_i,k_j}} = 1, \ \ \forall e_{ij} \in E_v^{new}, \ \forall i \in N_v^{old}, \ \forall j \in N_v^{new} \tag{4.4.8}$$

The case where destination extremity j of virtual link e_{ij} is an old node and source extremity i is a new node: Each new virtual link e_{ij}, having an old VNF j as destination endpoint, is mapped to exactly one physical path P_{k_i,m_j} where k_i is a physical candidate for the new VNF i and m_j is the initial mapping of the old VNF j.

$$\sum_{k_i \in \mathcal{C}} y_{e_{ij}, P_{k_i,m_j}} = 1, \ \ \forall e_{ij} \in E_v^{new}, \ \forall i \in N_v^{new}, \ \forall j \in N_v^{old} \tag{4.4.9}$$

Bandwidth constraint: Obviously the residual bandwidth in a physical path P_{k_i,k_j} has to satisfy the bandwidth requirement of virtual link e_{ij}. The allocation (or mapping) must not violate the remaining bandwidth of each physical link $e \in E_p$ on the physical path P_{k_i,k_j}. Note that we compute $\mathcal{K}$-shortest paths candidates P_{k_i,k_j} using Dijkstra's algorithm based on residual bandwidth of links. This condition can be formally expressed as:

$$\begin{aligned} \sum_{e_{ij} \in E_v^{new}} bw_{e_{ij}} \times y_{e_{ij}, P_{k_i,k_j}} \times \delta_{ep} \leq BW_e, \\ \forall e \in P_{k_i,k_j}, \ \forall P_{k_i,k_j} \in \mathcal{P} \end{aligned} \tag{4.4.10}$$

Node and link mapping constraint (source type): When a VNF i is mapped to a physical candidate node k_i, each virtual link e_{ij}, starting from VNF i, has to be mapped to a physical path P_{k_i,k_j} with k_i as one of its endpoint or extremity (source). The other endpoint is k_j.

$$\sum_{k_j \in \mathcal{C}} y_{e_{ij},P_{k_i,k_j}} = x_{ik_i}, \quad \forall e_{ij} \in E_v^{new}, \ \forall k_i \in \mathcal{C} \tag{4.4.11}$$

Node and link mapping constraint (destination type): Similarly to the previous constraint (source type), if a VNF j is mapped to a physical candidate node k_j, then each virtual link e_{ij} has to be mapped to a physical path having k_j as one of its endpoints (destination).

$$\sum_{k_i \in \mathcal{C}} y_{e_{ij},P_{k_i,k_j}} = x_{jk_j}, \quad \forall e_{ij} \in E_v^{new}, \ \forall k_j \in \mathcal{C} \tag{4.4.12}$$

Node separation constraint: This constraint corresponds to situations where VNFs have to be separated and mapped onto distinct nodes for security reasons for example or simply due to tenant requirements or application constraints. In this case VNF i and j are mapped into nodes k_i and k_j with $k_i \neq k_j$.

$$x_{ik} + x_{jk} \leq 1, \quad \forall i,j \in N_v^{new}, \ \forall k \in \mathcal{C} \tag{4.4.13}$$

Objective function: The objective function directly depends on the infrastructure providers' interests while taking into account as much as possible the users' or tenants' expectations. The objective function has to be adapted to the provider policies and criteria such as a consolidation, an energy consumption minimization or a cost reduction and revenue maximization. There are consequently multiple possible objective functions, but as far as this chapter is concerned, we will arbitrarily, and for the sake of selecting a given case, assume that the providers aim at naturally balancing the load on their hosting infrastructures. If the providers aim at reducing energy consumption, we would modify the objective function to achieve consolidation with a maximum packing of the requests and hence minimize energy consumption and cost, by for instance selecting the nodes whose residual capacity is sufficient and closest in size to the requests. A simple and efficient way to achieve load balancing is to favor (select in priority) the infrastructure nodes and links that are least loaded to host

the requests, resulting in gradual and distributed filling or loading of the infrastructure. The proposed objective function consists in maximizing Z of Equation 4.4.14.

$$Z = \sum_{i \in N_v^{new}} cpu_i \times (\sum_{k \in \mathcal{C}} \frac{CPU_k}{min_{CPU}} \times x_{ik}) \tag{4.4.14}$$

The variable min_{CPU} is the smallest residual compute power available in all the candidate nodes in set C. This is a normalization factor, used to align all terms in Equation 4.4.14 (dimensionless quantities). This factor is constant at each run of the ILP and it emphasizes in addition nodes with more free resources. The addressed VNF-FG extension problem is finally summarized by lumping the objective function and the constraints:

maximize $\{Z\}$

subject to:

$x_{im_i} = 1, \ \forall i \in N_v^{old}$

$y_{e_{ij},P_{m_i,m_j}} = 1, \ \forall e_{ij} \in E_v^{old}$

$\sum_{k \in \mathcal{C}} x_{ik} = 1, \ \forall i \in N_v^{new}$

$\sum_{i \in N_v^{new}} cpu_i \times x_{ik} \leq CPU_k, \ \forall k \in \mathcal{C}$

$\sum_{k_i \in \mathcal{C}} \sum_{k_j \in \mathcal{C}} y_{e_{ij},P_{k_i,k_j}} = 1, \ \forall e_{ij} \in E_v^{new}, \forall i,j \in N_v^{new}$

$\sum_{k_j \in \mathcal{C}} y_{e_{ij},P_{m_i,k_j}} = 1, \ \forall e_{ij} \in E_v^{new}, \forall i \in N_v^{old}, \forall j \in N_v^{new}$

$\sum_{k_i \in \mathcal{C}} y_{e_{ij},P_{k_i,m_j}} = 1, \ \forall e_{ij} \in E_v^{new}, \forall i \in N_v^{new}, \forall j \in N_v^{old}$

$\sum_{e_{ij} \in E_v^{new}} bw_{e_{ij}} \times y_{e_{ij},P_{k_i,k_j}} \times \delta_{ep} \leq BW_e, \ \forall e \in P_{k_i,k_j}, \forall P_{k_i,k_j} \in \mathcal{P}$

$\sum_{k_j \in \mathcal{C}} y_{e_{ij},P_{k_i,k_j}} = x_{ik_i}, \ \forall e_{ij} \in E_v^{new}, \forall k_i \in \mathcal{C}$

$\sum_{k_i \in \mathcal{C}} y_{e_{ij},P_{k_i,k_j}} = x_{jk_j}, \ \forall e_{ij} \in E_v^{new}, \forall k_j \in \mathcal{C}$

$x_{ik} + x_{jk} \leq 1, \ \forall i,j \in N_v^{new}, \forall k \in \mathcal{C}$

PROBLEM 2: VNF-FG extension optimization summary

4.4.1.1 ILP with Reduced Number of Candidate hosts (RNC_ILP)

The ILP is known not to scale polynomially with problem size, as the problem is NP-Hard [96], [3]. A way to reduce the complexity and to scale much better with size, is to explore not all the "candidate nodes and links" space and accept a suboptimal solution. That is cutting the exploration to a subset of the candidates, especially candidate nodes and links that are less loaded to favor load balancing. In order to avoid local optima, we use a random selection process to diversify choices and move out of local optima (traps) in the problem convex hull. This suboptimal selection process nevertheless takes into account the criteria and constraints expressed in Equation 4.4.14. The performance of this ILP inspired heuristic, using a reduced set of candidates, is compared with the ILP to assess the proximity in achieved solution quality or distance with the optimal solution from the full exploration ILP.

The pseudo-code of the candidate subset selection is depicted in Algorithm 2 where we check the initial mapping of the old VNF-FG graph G_v^{old} and avoid the locations used to host the old VNFs "$UsedNode()$" to favor load balancing. We randomly select eligible hosts "$randomNode$" from the "$queue$" to avoid being trapped in a local optimum. The "$queue$" is sorted by available resources.

Algorithm 2: Candidate selection module pseudo-code

```
Inputs: G_p, G_v^new, the initial mapping of G_v^old, maxCand
Output: A set of candidate hosts
queue ← ∅
C ← ∅
foreach physical node k ∈ N_p do
    if UsedNode(k) = False then
        queue ← queue ∪ k
stack ← Sort(queue)
repeat
    C ← C ∪ Pop(stack) ∪ randomNode
until size(C) = maxCand;
return C
```

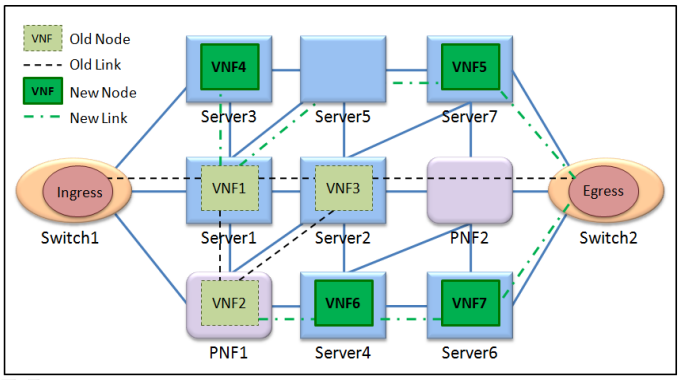


FIGURE 4.3: Example of the improved STVE algorithm

4.4.2 STVE: Steiner Tree based algorithm For VNF-FG Extension

An alternative to the ILP is to view the extension problem as a form of spanning tree problem where the solution for the extension has to cover a number of anchored nodes more precisely those of the previously deployed graph that have to be connected to the new graph (or extension). An equivalent way to address this optimization problem is to view the previous nodes as root nodes in a tree and the objective would be to find the branches and leaves out of these root nodes.

Figure 4.3 depicts a VNF-FG extension request to respond to rising requirements and demands. Server1, hosting VNF1, can be defined as a root tree-node and the objective would be to find a placement solution for the new VNFs, VNF4 and VNF5 part of the extension request, in leaf physical nodes. Similarly, PNF1 is seen as a root tree-node, since it already hosts VNF2 that must be connected to the new VNFs, VNF6 and VNF7. These new VNFs have to find a mapping solution in the ensuing tree leaves.

We nevertheless prefer to view the problem as a Steiner Tree optimization problem that is well known and fully characterized in combinatorial optimization. Indeed, determining a minimum cost connected network that spans a given set of vertices or nodes (known as terminals) is the well identified and studied spanning tree problem in the literature often solved using the cost of a minimum spanning tree (MST)[97] or solved by computing a Steiner Tree of minimal cost [98], [99], [100] and [101].

If a Steiner tree problem in graphs contains exactly two terminals, it reduces to finding the shortest path. If, on the other hand, all vertices are terminals, the Steiner tree problem in

graphs is equivalent to the minimum spanning tree. However, while both the shortest path and the minimum spanning tree problem are solvable in polynomial time, the Steiner tree problem in graphs is NP-complete [102].

The Steiner tree problem in graphs can be seen as a generalization of the shortest path problem (when the problem contains exactly two terminals) and the minimum spanning tree problem (when all vertices are terminals). In the addressed VNF-FG extension problem, the nodes of the previously deployed graph, that the extension graph nodes are connected to, correspond to Steiner Terminals. The goal is to find Steiner Vertices acting as intermediate nodes and hosts of the requested extension graph VNFs. To construct an efficient tree solution to host the new VNFs and place the extension graph while respecting imposed connectivity with the old graph, we use the minimum Steiner Tree approach. The tree cost is defined as the total cost of the links forming the tree.

The problem of computing a minimum cost tree for a given new graph with source s and set of physical nodes candidates $\mathcal{C}$ is modeled as a Steiner tree problem and defined as follows: Given a substrate graph $G_p = (N_p, E_p)$ with a cost associated with each physical link and a subset of the vertices $X \subseteq N_p$, the goal is to find a minimum cost tree that includes all the nodes in X. For finding the minimum cost tree, we set $X = \mathcal{C} \cup \{s\}$ (i.e., source s should have a path to every node in $\mathcal{C}$).

Since in the context of the VNF-FG extension, the substrate network links are undirected, the problem corresponds to the undirected Steiner tree problem found in for instance [103] and [104]. The Steiner tree problem is NP-complete [102] and it is therefore very unlikely that a polynomial algorithm for the Steiner tree problem exists. Several exponential enumeration algorithms have been suggested [101]. Due to the inherent difficulty in solving the Steiner tree problem, much effort has been devoted to the development of approximation algorithms which produce good quality solutions.

An integrative overview of the algorithmic characteristics of three well-known polynomial time heuristics for the undirected Steiner minimum tree problem: shortest path heuristic (SPH), distance network heuristic (DNH), and average distance heuristic (ADH) is given in

[105]. The worst-case time complexity of the SPH, DNH, and ADH heuristics is respectively $O(pn^2)$, $O(m + nlogn)$, and $O(n^3)$ where p is the number of vertices to be spanned, n is the total number of vertices, and m is the total number of edges [105].

In this chapter, we make use of the Shortest Path Heuristic (SPH), originally developed in [106], because of its bounded performance (gap with optimal) and its acceptable complexity (worst-case time complexity of the SPH is $O(pn^2)$). As shown in [106] the worst-case error ratio between the solution T obtained by the SPH and the optimal solution T_o is never greater than 2. More specifically,

$$\frac{c(T)}{c(T_o)} \leq 2 \times \left(1 - \frac{1}{p}\right) \tag{4.4.15}$$

where c is the edge-cost function and p is the number of vertices to be spanned. Furthermore, this bound is tight in the sense that there are problem instances for which the ratio equals the bound.

4.4.2.1 Links Costs for Steiner Tree

The links costs for this computation are determined using the critical link concept to achieve the VNF-FG extension. We define a substrate link to be critical with respect to a physical source s and a set of candidates $\mathcal{C}$ if this link has the highest cost (i.e., very loaded link with few available bandwidth capacity). Once the critical links are identified, we avoid hosting the new virtual links on critical substrate links as much as possible. The link cost is defined in term of the residual bandwidth in the substrate link. These weights are used to select in priority the less critical links (less loaded links) and the nodes with highest available residual capacity. The residual CPU capacities on the physical nodes are also taken into account in the selection.

4.4.2.2 Steiner Tree based algorithm for VNF-FG Extension (STVE)

In the existing literature, many mechanisms can be used to calculate the distances from the source s to a set of physical candidate nodes $\mathcal{C}$ (e.g., Dijkstra's shortest path algorithm, Min-hop shortest path tree (MHT), etc). We choose the Min-hop shortest path with respect to the nodes' and links' capacities and costs to build our Steiner Tree based heuristic algorithm for VNF-FG extension placement and chaining. We run this algorithm starting from a source node s (i.e., s can be a substrate location of an old virtual node) until we reach a node candidate $k \in \mathcal{C}$. Selecting lower cost links and nodes with highest leftover capacity along the paths leads to the nodes k that will host the VNFs in the extension graph. These paths are added to the Steiner tree solution.

The proposed STVE algorithm operates in 5 steps:

1. Check the initial mapping of old VNF-FG graph G_v^{old};
2. Decompose the G_v^{new} to find an efficient mapping and construct the tree solution sequentially;
3. Find a set of physical candidate nodes $\mathcal{C}$ related to a fixed source s that promote the using of lower cost links (Method $GetCandidates$): if node k available capacity and type constraints are met, it can be a candidate;
4. Compute the shortest path between source s and candidate k using Min-hop-shortest-path algorithm;
5. Check link mapping: if links capacities are respected, confirm candidate k as a mapping solution for new VNF and add the solution path from s to k to the Steiner tree built so far;

The pseudo-code of this heuristic is shown in Algorithm 3.

4.4.3 EDVE: Eigendecomposition For VNF-FG Extension

An alternative approach is to address the VNF-FG extension problem by extending and adapting the eigendecomposition method for VNF Placement and Chaining described in

Algorithm 3: STVE algorithm pseudo-code

```
Inputs: G_p = (N_p, E_p) with edge costs, G_v^new = (N_v^new, E_v^new), the initial mapping of
  old VNF-FG graph G_v^old = (N_v^old, E_v^old)
Output:A low-cost Steiner trees hosting the VNF-FG extension graph G_v^new
foreach new virtual link e_ij ∈ E_v^new do
    if (VNF i or VNF j) ∈ N_v^old then
        s ← GetInitialMapping(VNF i or VNF j)
        C ← GetCandidates(s)
        stack ← Sort(C)
        while stack ≠ ∅ and solution = False do
            k ← Pop(stack)
            if Min-hop-shortest-path(s, k) = True then
                solution ← True
                ST ← ST + GetPathSolution(s, k)
    else if (VNF i and VNF j) ∈ N_v^new then
        s ← GetSolutionExtensionMapping(VNF i or VNF j)
        C ← GetCandidates(s)
        stack ← Sort(C)
        while stack ≠ ∅ and solution = False do
            k ← Pop(stack)
            if Min-hop-shortest-path(s, k) = True then
                solution ← True
                ST ← ST + GetPathSolution(s, k)
return ST
```

[107]. This algorithm uses the Umeyama's eigendecomposition approach [108] for optimal matching between weighted graphs. This state of the art method is based on the eigendecomposition of the adjacency matrices of the virtual graph and the physical graph (graph to host the request) and on the Hungarian method [109] to extract mapping results.

4.4.3.1 Eigendecomposition for VNF Placement and Chaining

The proposed method in [107] casts the VNF Placement and Chaining problem in networked cloud infrastructures into the weighted graph matching problem (WGMP) that finds a one to one mapping function ϕ between N_p and N_v that minimizes a distance criterion between G_p (the substrate graph) and NCT_v (requested virtual graph derived from the VNF-FG requests G_v).

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

Since the eigendecomposition deals with graphs of the same size, the proposed algorithm in [107] adds dummy isolated vertices to the request graph in order to make the graphs of equal size. The algorithm, in fact, adds rows and columns with zeros to the adjacency matrix A_{NCT_v} to line the size to that of the substrate graph (hosting infrastructure).

Starting from the extended adjacency matrix A_{NCT_v} (padded with zeros) and the adjacency matrix of the substrate graph A_{G_p}, the algorithm derives the eigenvectors of these adjacency matrices, U_{G_p} and U_{NCT_v}. The algorithm builds from these eigenvectors a matrix M and a permutation matrix P that contains a 1 in positions corresponding to the VNFs mapping results.

4.4.3.2 Eigendecomposition For VNF-FG Extension (EDVE)

To address the VNF-FG extension problem we also make use of our Eigendecomposition algorithm [107] by extending and adapting it to connect a new request to a previously deployed slice or forwarding graph. The initial deployment is obtained using also the Eigendecomposition so the initial adjacency matrices, traces and permutation matrices are produced and available for the extension when such a request is expressed by the tenant. The extension is realized by considering the new VNF-FG graph G_v^{new} (formulated as a request graph NCT_v^{new}) and the updated substrate graph G_p to find efficient locations for new VNFs. Before searching a mapping for the VNF-FG extension, our proposed algorithm updates the adjacency matrix of the substrate graph A_{G_p} by:

- anchoring the initial mapping NCT_v^{old} by removing reserved (allocated) resources (CPU on nodes and bandwidth on links) for these previously deployed VNFs and fixing (freezing) the location in matrix M and thus in permutation matrix P of the previously deployed nodes and of course the key nodes (in this set) to which some of the extension graph nodes have to be connected to;

- setting deliberately the capacity of some of the used physical nodes to zero, to avoid co-localization of two VNFs in the same substrate node. This also ensures load balanced Eigendecomposition algorithm solutions.

The adjacency matrix of the new VNF-FG Extension request is the second matrix used by the Eigendecomposition algorithm to achieve the mapping of the new graph on the hosting infrastructure while respecting connectivity of the extension with the old graph. This matrix contains the new nodes composing the VNF-FG extension request with the nodes connected to the old graph (previously deployed tenant slice or VNF-FG) especially tagged to achieve the necessary constrained placement. This constrained placement consists in finding the links and paths that will interconnect these new nodes to their corresponding nodes in the previous graph to respect the required inter-graph connections. The other VNF-FG extension nodes are unconstrained and placed as usual according to the selected (desired) optimization criterion.

4.5 Performance Evaluation

The algorithms are compared using extensive simulations (conducted on an experimental cloud and networking platform comprised of physical servers and networking technologies, similarly to our previous work in [107] and [110]) where both the requests and hosting infrastructures are drawn using standard graph generation tools (such as the GT-ITM Tool [86]) and real infrastructure topologies (such as the Germany50 network topology [85]). The comparison includes the ILP algorithm, the reduced candidate set ILP algoithm, the Steiner Tree based algorithm (STVE), the modified Eigendecomposition algorithm as proposed in this work (EDVE) as well as the basic Eigendecomposition algorithm applied to the previous and new graph extension together by combining the two graphs into a common composite graph to be mapped by the unmodified Eigendecomposition (called "ReassignAll"). This unmodified version of the Eigendecomposition is only used to assess how suboptimal the modified Eigendecomposition is. We obviously, do not advocate the use as is of the unmodified Eigendecomposition [107] on the composite graph (i.e., old and extension merged into one composite graph to be mapped at once) since this would require

migration of the networking services or functions and thus interruption of services associated to the previously deployed graph. This version provides nevertheless a means to assess deviation from best achievable performance by the Eigendecomposition.

4.5.1 Simulation Environment

All simulations for all the algorithms run in a dedicated server in the experimental platform with the following processing capabilities: a 2.50 GHz, Quad Core server with 6 GBytes of available RAM. The VNF-FG requests are generated using a Poisson process with an average arrival rate of 2 requests per 100 time units. The lifetime of each request follows an exponential distribution with a mean of 50 time units. The Germany50 network topology [85] is used for the first assessment (with 50 nodes). This topology is defined by the German National Research and Education Network (DFN). The capacity of physical nodes and links are generated randomly in the $[100, 120]$ interval. The size of the initial VNF-FG requests is arbitrarily set to 4 nodes and the size of the extension is set to 3 nodes for each VNF-FG request. The GT-ITM [86] tool is used to generate the requested VNF-FG topologies. The required capacities of the initial VNF-FG are fixed to 10 CPU units for each virtual node and 10 bandwidth units for each virtual link. The extended VNF-FG computing and bandwidth requirements are set to 20 CPU units per node and 20 bandwidth units per link. The connectivity between nodes in the VNF-FG is set to 0.3 (30%).

The performance assessment at larger scale uses the GT-ITM tool to generate network topologies with 200 nodes and a connectivity of 0.3 (or 30%). The physical resource capacities (i.e., CPU and bandwidth) are also drawn randomly in the $[40, 50]$ interval. The VNF-FG requests are generated using a Poisson process with an average arrival rate of 5 requests per 100 time units. The lifetime of each request follows an exponential distribution with a mean of 200 time units. The size of the initial VNF-FG requests is also set to 4 nodes and the size of the extension is set to 3 nodes for each VNF-FG request. The required CPU for each VNF in the initial VNF-FG is set to 15 units. The required bandwidth between two VNFs, to ensure communication between them, is set also to 15 units. The extension VNF-FG computing and bandwidth requirements are set to 20. The connectivity between nodes in the VNF-FG is set to 0.3 (30%).

For the realistic topology and the large-scale evaluations, 1000 VNF-FG requests are generated and an extension request is triggered for each generated request. Since we are focussing on extension and adaptation of already embedded VNF-FGs, without any loss in generality, we used the heuristic approach of [107] to generate the initial VNF-FG mapping. The ILP solver used in our experiments is Cplex [111].

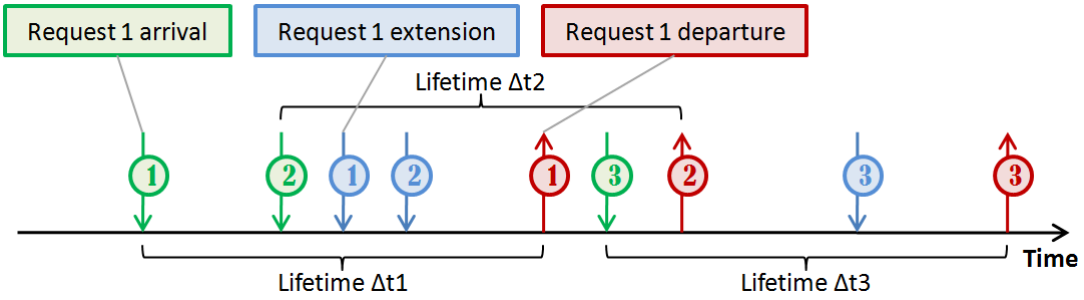


FIGURE 4.4: Processing of VNF-FGs

Arrivals, extensions, and departures occur over time as depicted in the example in Figure 4.4. The time an extension request might take to arrive is a random variable between the arrival and departure time of the corresponding initial request.

4.5.2 Performance Metrics

The metrics used for the performance evaluation are typical indicators for placement algorithms and as defined below:

- **Successful extensions** is the number of VNF-FG extension requests that are fulfilled or successfully placed in the hosting infrastructure while respecting connectivity constraints with the previously deployed graph. This quantity should be maximized since it is equivalent to minimizing rejection rate of requests for the algorithms.

- **Extension ratio** is the ratio of successful extensions to the number of initial VNF-FG graph requests that have been accepted. Since the initial VNF-FG placement is achieved using the Eigendecomposition algorithm to initialize the evaluation with the initial graph before extension for all the algorithms, this metric reflects the relative efficiency of the algorithms, starting from the very same previously deployed graph for all algorithms to compare them on a common and fair basis.

- **Execution time** is the time required by each algorithm to find a placement solution for the extension requests. This metric reflects the algorithms complexity.

- **Acceptance revenue** is the service provider realized (generated) revenue (benefits) at time t when extension requests are successfully fulfilled and accepted. The acceptance revenue is formally expressed as:

$$\mathbb{R}(t) = \sum_{\mathcal{R}eq_{ex} \in \mathcal{AR}_t} \mathbb{R}\left(\mathcal{R}eq_{ex}\right) \tag{4.5.16}$$

where $\mathcal{AR}_t$ is the set of accepted extension requests up to time t and $\mathbb{R}(\mathcal{R}eq_{ex})$ is the extension request $\mathcal{R}eq_{ex}$ gain expressed as in [112] and [113]:

$$\mathbb{R}(\mathcal{R}eq_{ex}) = \alpha \sum_{i \in N_v^{new}} (cpu_i \times U_{cpu}) + \sum_{e_{ij} \in E_v^{new}} \left(bw_{e_{ij}} \times U_{bw}\right) \tag{4.5.17}$$

where U_{cpu} is the realized revenue from allocating one unit of CPU resources and U_{bw} is the earned revenue from the allocation of one bandwidth unit, and $\alpha = \sum_{k \in \mathcal{S}} \frac{CPU_k}{min_{CPU}}$ is a tunable weight that reflects the quality of the solutions. In fact, the revenue increases by choosing the best locations which correspond to nodes with more free resources (i.e., $\mathcal{S}$ is the set of physical nodes solutions, the other parameters CPU_k and min_{CPU} are described in Section 4.4).

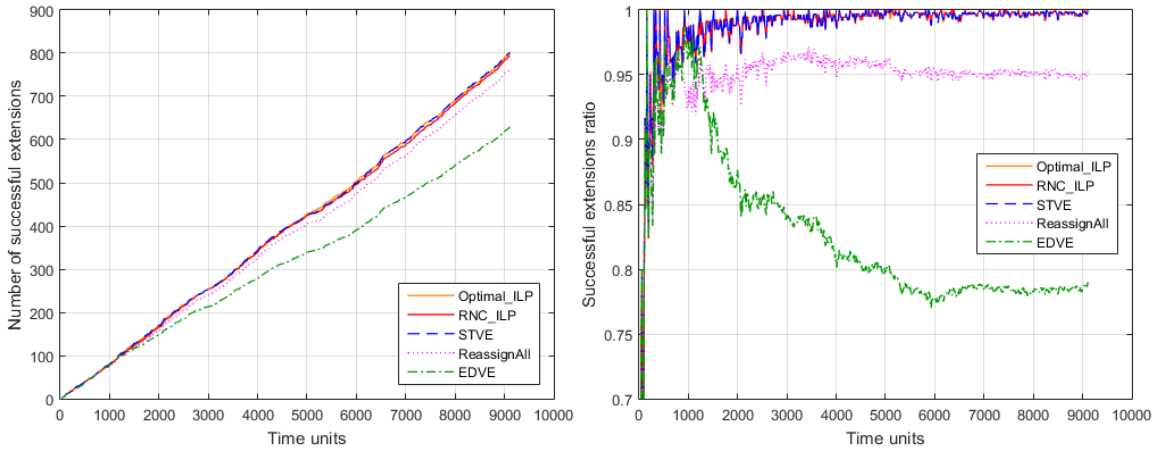
Note that $\mathbb{R}$ is the total revenue taking into consideration all the incoming extension requests.

4.5.3 Evaluation Results

The performance is evaluated on the Germany50 network for all algorithms since the size of this topology is small and the ILP algorithm produces solutions in acceptable time. For larger graph sizes, we resort to random graph generations with hundreds of nodes and links and compare the performance of all other algorithms without the ILP that does not scale with problem size.

4.5.3.1 Evaluation on a realistic topology

The topology used for the evaluation is as stated the Germany50 network with a topology of 50 nodes. The ILP is included in the performance comparison for this topology. Since we also use a modified ILP, a heuristic called the RNC_ILP, that uses a subset of candidates and hence uses partial exploration, we denote the original and optimal ILP as "Optimal" to differentiate explicitly these algorithms and avoid any confusion.



(a) Successful extensions (b) Extensions ratio

FIGURE 4.5: Germany50 network topology results: Extensions

Figure 4.5(a) and Figure 4.5(b) depict respectively the results in terms of successful extensions and extension ratio (two similar metrics with the second normalized metric by a common reference graph, the initially deployed slice or VNF-FG prior to the extension). The Optimal_ILP, the RNC_ILP and the STVE algorithms outperform the eigendecomposition based solutions (i.e., EDVE, ReassignAll) by accepting 800 extension requests versus 629 for the EDVE. The normalized metric confirms these results passed a transient state (passed 2000 time units) where the achieved ratios are around 99% for the Optimal_ILP, the STVE and the RNC_ILP, 95% for the ReassignAll, and only 78% for the EDVE. The Eigendecomposition approach is known to provide much better results for highly connected topology compared with weakly connected graphs as will be shown in the more extensive simulations with randomly generated topologies.

Figures 4.6 and 4.7 compare the quality realized by the proposed algorithms in terms of objective function, achieved maximum in Equation 4.4.14, to provide insight on the relative

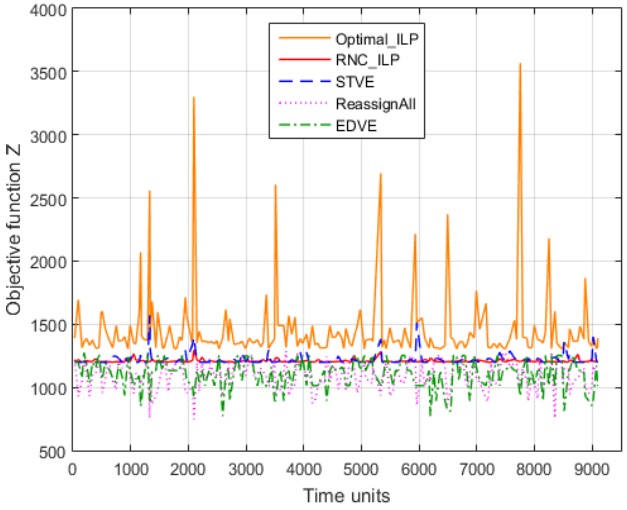


FIGURE 4.6: Quality of the mapping

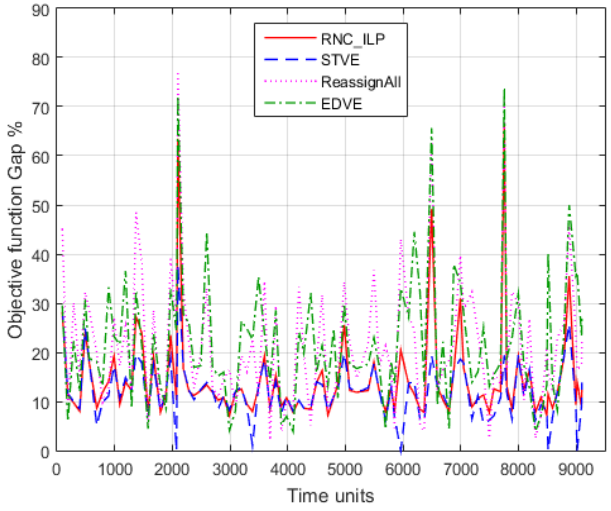


FIGURE 4.7: Objective function gap compared with Optimal_ILP

performance of the heuristics to the optimal solutions found by the ILP. Since we aim in this chapter at maximizing provider revenue, we define the mapping quality based on the value of the objective function Z that describes the algorithms ability to achieve load balancing on the hosting infrastructure.

The gap is computed using Equation 4.5.18.

$$Gap(\%) = \frac{Z\left(Optimal\_ILP\right) - Z\left(ALG\right)}{Z\left(Optimal\_ILP\right)} \times 100 \tag{4.5.18}$$

where ALG represents the selected algorithm for comparison with the ILP, that is: EDVE, ReassignAll, RNC_ILP or STVE.

Figure 4.6 reports the achieved Z values and indicates clearly that the closest in performance to the ILP is the Steiner algorithm (STVE). The other algorithms are outperformed in terms of quality of the solutions. Figure 4.7 depicts the gap of the algorithms in quality of the solutions compared with the ILP, and reveals in an enhanced scale their relative performance. The STVE is shown to be within 10% to 20% of the ILP performance and in some cases achieves the same maximum for the objective function (points touching the abscissa at points 2073, 5961, 8515, 9018 time units).

The next closest algorithm in achieved quality is the RNC_ILP whose gap lies in the range 10% to 30% of the ILP objective function. The Eigendecomposition approaches, ReassignAll and EDVE, are further away in quality from the ILP and the gap can exceed in some cases 70% and spans typically a larger range from 10% to 78%. Obviously ReassignAll, that embeds again all the VNF-FGs (old and new) outperforms the EDVE that embeds only the extension without affecting the previously deployed VNF-FG or service graph.

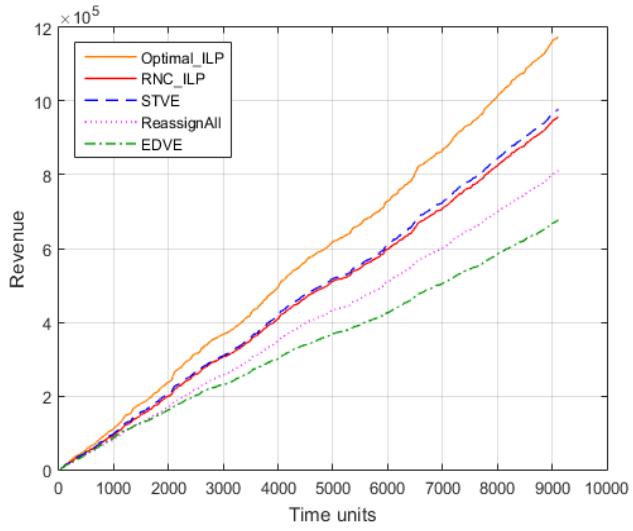


FIGURE 4.8: Acceptance revenue

The computed acceptance revenue reported in (Figure 4.8) corresponds to the total revenue generated when accepting the new extension requests at time t. The Optimal_ILP achieved provider revenue is 16.75% higher than the STVE, and respectively 30.94% and 42.3% better than ReassignAll and EDVE. This interpretation confirms the results obtained in

Figure 4.5 on successful extensions. Summarizing all these results, the STVE stands out as the closest in performance to the ILP and it will be shown that this algorithm scales best with increasing problem size if larger graphs are generated for the performance evaluation.

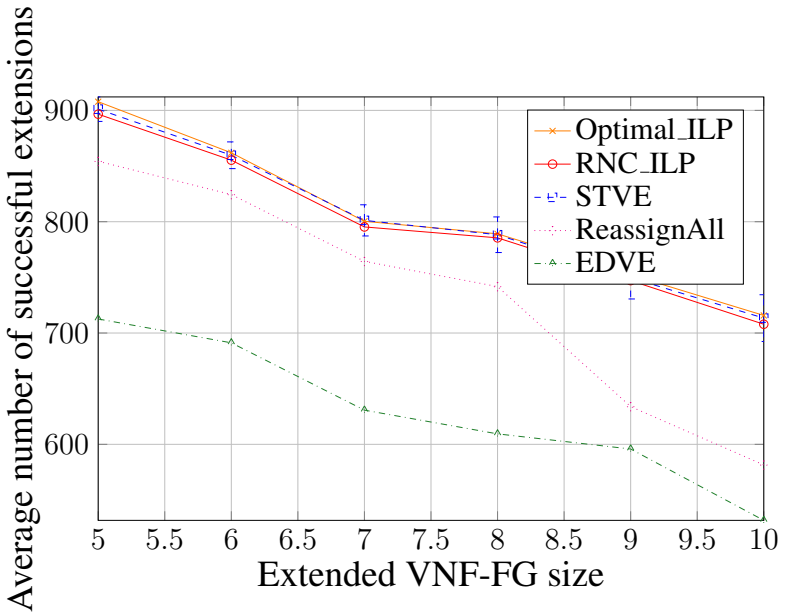
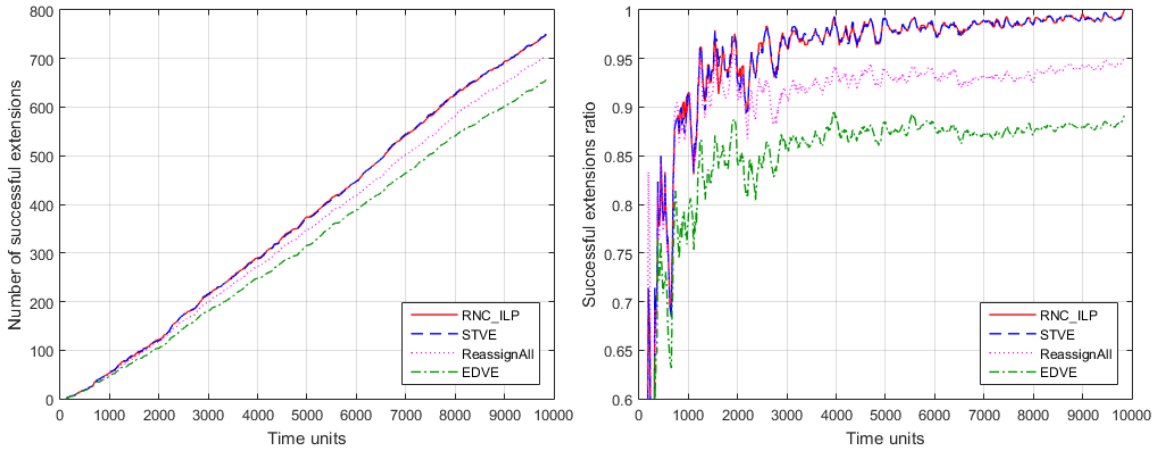


FIGURE 4.9: Successful extensions w.r.t VNF-FG size variation

The results in Figure 4.5 to Figure 4.8 reflect the observed behavior of the algorithms as a function of simulation time over multiple runs. Figure 4.9 evaluates the average number of successful extensions of the algorithms as a function of increasing VNF-FG request sizes while varying the capacity of nodes and links. The results in Figure 4.9, presented with a 95% confidence interval (shown only for the Steiner algorithm for legibility reasons), depict a high similarity with results in Figure 4.5(a) (more precisely for VNF-FG size of 7 nodes).

4.5.3.2 Large-scale evaluation

To analyze the behavior of the algorithms and their scalability with problem size, larger hosting infrastructures (NFV-Is with 200 nodes) are randomly generated using the GT-ITM Tool. This evaluation should confirm the previous results and reveal the ability of the algorithms to scale with increasing problem size. In addition to the previous performance metrics, we report the execution time of each algorithm to shed light on their scalability.



(a) Successful extensions (b) Extensions ratio

FIGURE 4.10: Large-scale network topology extension results

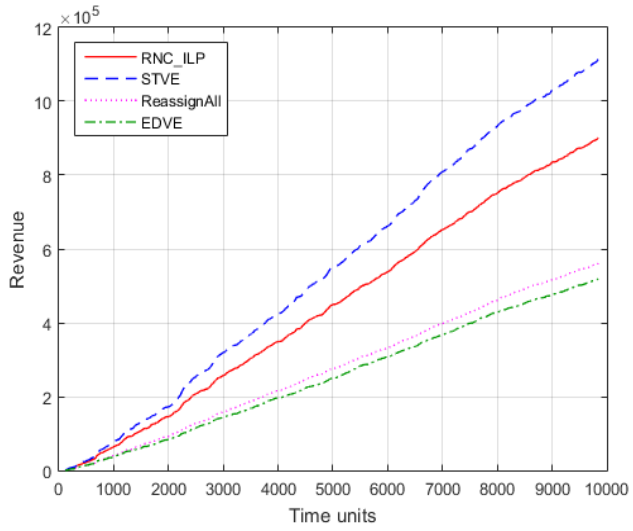


FIGURE 4.11: Acceptance revenue

Figures 4.10(a), 4.10(b), and Figure 4.11 report similar results as for the Germany50 topology. We notice that the RNC_ILP and the STVE algorithms outperform the eigendecomposition based solutions (i.e., EDVE, ReassignAll) by accepting more requests and ensuring higher revenues.

Figure 4.12 depicts the average number of successful extensions as a function of NFV-I size (varying from 100 to 500). Note that several network topologies are considered for each fixed number of physical nodes to ensure that our results are stable and reliable. The results obtained from multiple simulations are averaged and presented with 95% confidence interval on the reported results (shown only for the Steiner algorithm for legibility reasons).

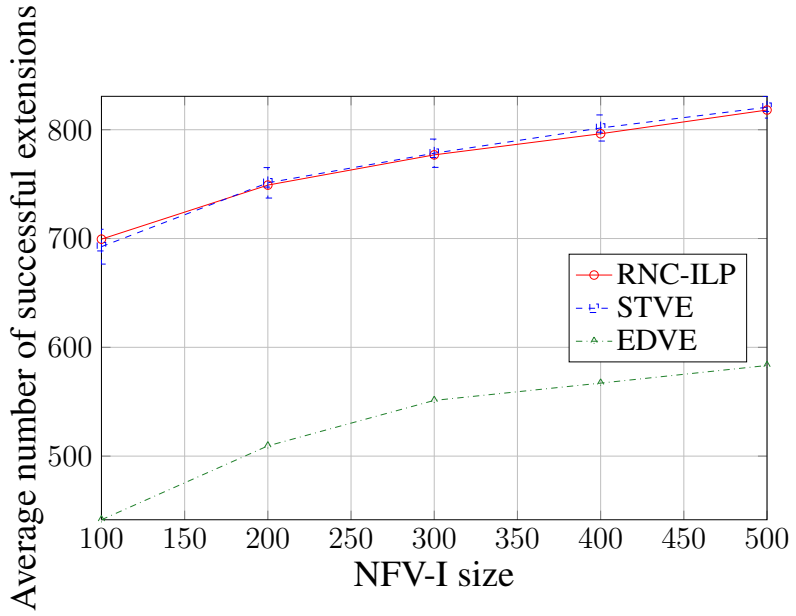


FIGURE 4.12: Successful extensions w.r.t NFV-I size variation

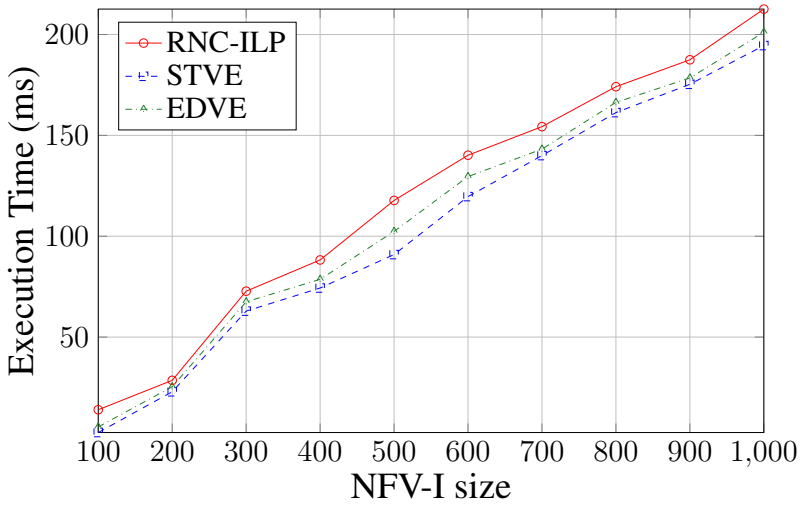


FIGURE 4.13: Execution Time w.r.t NFV-I size variation

4.5.3.3 Execution time (Convergence time)

The aspect that deserves more attention is the algorithms execution time as reported in Figure 4.13 that clearly shows that the Steiner and Eigendecomposition methods have the best performance for this metric since they find solutions for the extension requests placement in few milliseconds for 100 nodes, few tens of milliseconds for 200 nodes and need no more than 60 milliseconds for 300 nodes. In addition, results reported in Figure 4.13 indicate that

the execution times of the Steiner and Eigendecomposition approaches grow linearly like the RNC_ILP. The RNC_ILP has a performance in the vicinity of these two algorithms but requires selection of a reduced set of candidates to realize this gain. Limiting the exploration of the ILP is not necessarily recommended if alternatives, such as the Steiner Tree based algorithm, can find better solutions in a shorter execution time. The eigendecomposition, even if faster, is outperformed by the RNC_ILP in terms of acceptance ratio, revenue, and proximity to the Optimal_ILP. An aspect that also requires additional investigation is the performance of the Eigendecomposition methods for highly interconnected topologies. Eigendecomposition algorithms are known to perform much better when connectivity is higher.

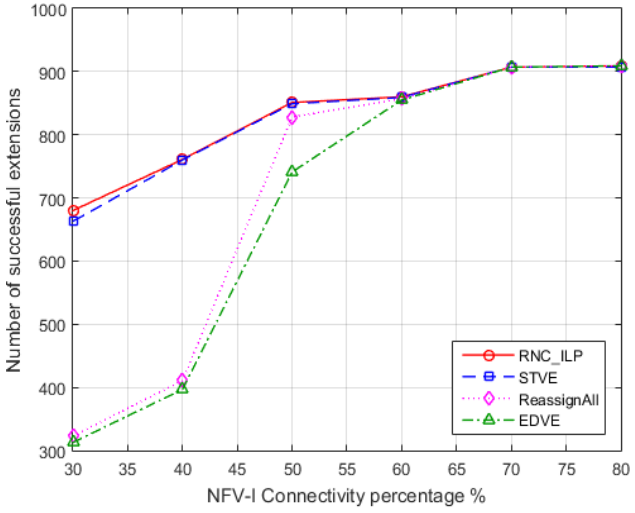


FIGURE 4.14: Successful extensions for varying NFV-I connectivity

4.5.3.4 Variation of NFV-I connectivity

To see the impact of the substrate graph connectivity on the behavior of our algorithms, the results are reported for an NFV-I connectivity percentage spanning a 30% to 80% connectivity in the evaluated hosting infrastructures with a number of nodes fixed at 100 nodes. The substrate graph size is fixed to 100 nodes. As previously hinted, the Eigendecomposition algorithms catch up in performance with all other algorithms as the network connectivity increases, especially when connectivity exceeds 60% as depicted in Figure 4.14. Figure 4.15 confirms the stability of the Eigendecomposition approaches with respect to execution time that remains fairly constant and more importantly rather low compared to the RNC_ILP

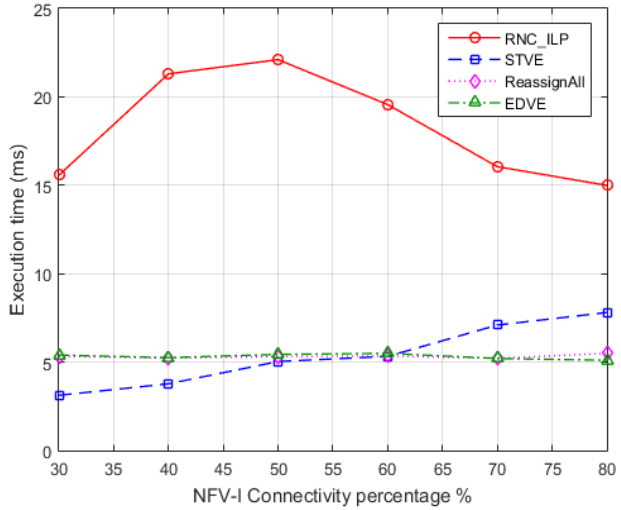


FIGURE 4.15: Execution time when varying the NFV-I connectivity

and very close to the Steiner based approach. The Steiner tends to increase slightly with increasing connectivity but this can be considered marginal with both the Steiner and Eigen-decomposition requiring around 5 milliseconds to find a placement solution for the extension requests. The RNC_ILP takes longer, in the order of tens of milliseconds ($> 15ms$) to find a solution for the evaluated graph sizes (NFV-Is with 100 nodes).

4.6 Conclusion

This chapter, to the best of our knowledge, is probably the first to address the problem of VNF-FG extension with the main objective of maximizing provider revenue by reducing the rejection of extension requests when faced with dynamic and rising demand and traffic load. We propose an ILP model as an exact algorithm acting as a baseline for comparison and as the solution for small problem sizes. To improve scalability, we also propose two heuristic algorithms: a new Steiner Tree based algorithm and an eigendecomposition based approach. We show that the solutions can efficiently extend virtualized network functions forwarding graphs with good execution time and extension requests acceptation performance over distributed and dynamically varying cloud environments. The Steiner Tree is found to be the best overall performance tradeoff solution for the VNF-FG extension problem as it performs uniformly well across all scenarios and can scale for large problem sizes. The Eigendecomposition methods are also quite fast but only recommended if the hosting

infrastructures are highly connected, the only situation where they achieve similar performance to the Steiner and the ILP based solutions.