Enhanced Reinforcement Learning Approach for VNF-FG Embedding

5.1 Introduction

This chapter, as the previous ones, addresses also the placement of service function chains (SFC) and the placement of virtualized network function forwarding graphs (VNF-FG) in NFV Infrastructures (NFVIs). This NP-Hard placement problem has been extensively investigated in the past, just like in our previous contribution, by mostly achieving instantaneous and sequential placement of the requests. These approaches unfortunately result in optimal placement without consideration for longer term optimization and reward. The algorithms have seldom been combined with past knowledge (using learning) or with prediction of future demands to use resources more efficiently and increase the number of accepted requests. One way to realize this longer term reward is to model and resolve the problem using a Markov Decision Process (MDP) based characterisation. A practical way to resolve the longer term reward placement problem is to use Reinforcement Learning (combined with the MDP modeling or alone) instead of traditional combinatorial and convex optimization leading only to instantaneous optimality and to rather limited long term benefits (or reward).

To highlight the benefits of using a reinforcement learning approach for the VNF-FG and SFC placement problem, we compare the performance to the algorithms proposed in the previous chapters, namely Integer Linear Program (ILP) using an exact formulation for instantaneous optimal placement handling one request at a time. We use a load balancing criterion throughout the infrastructure for all algorithms involved in the performance comparison. Other criteria than load balancing can be selected, such as maximum packing and consolidation to reduce resource usage and favor minimization of energy consumption and operational cost. The relative performance of the algorithms is not expected to change or will not be affected if another criterion is adopted to conduct the comparison. We, hence, limit the study to load balancing objectives. The ILP, that does not scale with problem size, is used as a reference for small hosting infrastructures.

A modified ILP, that operates on a reduced set of candidate hosts, called R_ILP, is proposed to improve scalability and speed compared to the ILP and is part of the overall performance evaluation and assessment. A batch algorithm, called BR_ILP, that operates on a group of requests, handled as a composite graph request, is also included in the comparison since it is expected to perform better than the ILP in terms of placement efficiency over longer time scales. The performance of the algorithms is also compared with an Eigendecomposition algorithm with reasonable complexity (complexity in $O(n^5)$, where n is the hosting graph size) shown to have good placement performance. The Reinforcement Learning solution uses standard Q-Learning enhanced with expert knowledge to accelerate learning during training and when unknown conditions and situations emerge.

Section 5.2 presents related work. The problem formulation is described in Section 5.3. The proposed chain placement algorithms are introduced in Section 5.4. Section 5.5 reports the performance evaluation results and Section 5.6 summarizes the main findings.

5.2 Related work

The need to dynamically deploy virtualized network services on-demand, through VNF-FG embedding, is identified as the core technology of 5G networks. Therefore, this issue has been at the very center of academic and industrial research in recent years. Here, we give a summary of the use of Reinforcement Learning in networking and especially in the VNF-FG embedding domain.

Since the VNF-FG embedding problem can be well described as a Markov decision process (MDP), then Reinforcement Learning (RL) is a good framework to use to find approximate optimal solutions. Recently, some works proposed RL-based approaches for VNF-FG embedding. With RL, an agent learns by interacting with its environment. The agent learns to perform the best action for each state by performing actions and observing the rewards. Given enough observations, an optimal policy can be learned. Thus, the training data in reinforcement learning is a set of state-action-rewards. Authors in [114] map the VNF-FG in two stages: node mapping stage then link mapping stage. They propose two algorithms for link mapping, based on a Multi-Commodity Flow approach and a shortest-path approach. Regarding the node mapping, the authors propose an MDP-based approach. Consequently, this approach is time consuming and not adapted for real-time embedding. In [115], authors propose a multi-agent-based reinforcement learning approach for virtual network embedding. These agents evaluate the feedback to learn the best policy to adopt in order to optimally allocate the required resources to the virtual nodes and links. In [116], authors propose a deep reinforcement learning-based approach for multi-domain VNF-FG embedding. Authors in [117] tackle the SFC allocation problem and present a learning method that places VNFs on an appropriate node that maximizes the performance of VNFs according to the load condition of the physical network. However, this method takes a huge amount of time to converge under an extensive exploration space. In [118], authors propose an RL-based algorithm to solve the NP-hard VNF scheduling problem.

In some existing studies, the reinforcement learning is also used for VNF migration [78], [77], [119], and scaling [79], [72] problem under dynamic network load. Furthermore, the large action space, when selecting which VNF instance to migrate, leads to high complexity and poor convergence performance.

To the best of our knowledge, we are the first to propose an enhanced RL-based approach combined with an expert knowledge mechanism to avoid a lengthy training process for VNF-FG placement and chaining.

5.3 Problem Description

The VNF-FG embedding optimization problem, extensively addressed in the literature, corresponds to the placement of the requested VNFs and flow paths in the hosting infrastructure.

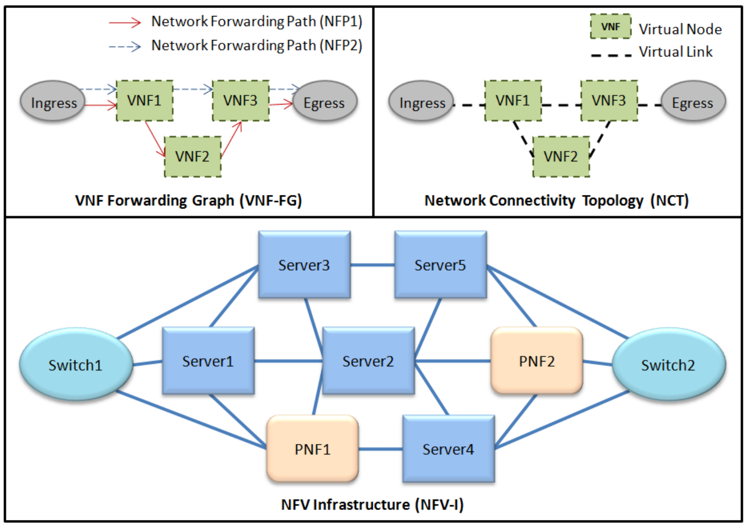
Network Forwarding Path (NFP1)
Network Forwarding Path (NFP2)
Ingress
VNF1
VNF3
Egress
VNF2
VNF Forwarding Graph (VNF-FG)
VNF
Virtual Node
Virtual Link
Ingress
VNF1
VNF3
Egress
VNF2
Network Connectivity Topology (NCT)
Server3
Server5
Switch1
Server1
Server2
PNF2
Switch2
PNF1
Server4
NFV Infrastructure (NFV-I)

FIGURE 5.1: An example of VNF-FG and NFV-I topologies

5.3.1 Substrate Graph or NFV Infrastructure

Deriving an ILP formulation of the VNF-FG placement problem requires a mathematical graph representation of the service graph (SFC or VNF-FG) and the hosting infrastructure.

The physical infrastructure, 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 CPU_k (this can be easily extended to other types of resources: such as storage or memory capacity), and ii) type T_k: switch, server or Physical Network Function (PNF). 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 (serving the ingress and egress flows in the VNF-FG topologies) and some interconnected servers is presented in Figure 5.1.

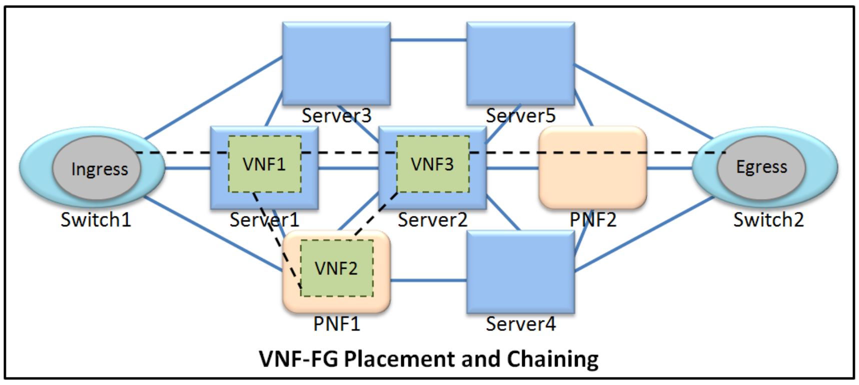

FIGURE 5.2: Mapping of the VNF-FG in the NFV-I

5.3.2 VNF Forwarding Graph or SFC Graph

The client request (i.e., a requested 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 $bw_{e_{ij}}$. Note that we can also consider instead the end to end latency if this is the criterion to be taken into account. Our model is generic and can be adjusted based on the client criteria and requirements.

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, NAT, etc.). 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 5.1 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.

5.3.3 VNF-FG placement and chaining

Figure 5.2 depicts a placement solution for the VNF forwarding graph (VNF-FG) highlighted by the dashed lines, starting from the ingress switch 1, crossing the VNF 1, VNF 2, and VNF 3 (hosted respectively in Server 1, PNF 1, and Server 2), and ending at the egress switch 2.

The infrastructure providers can map the VNFs using multiple objectives: such as minimizing mapping costs [120], maximizing acceptance ratios and improving provider gains, and improving energy efficiency [121].

5.4 Proposals

To solve the VNF-FG placement and chaining problem, we propose an ILP model and an enhanced Q-Learning based approach, and evaluate and compare their performance.

5.4.1 ILP Formulation

We now formulate the ILP model of the VNF placement and chaining problem using an objective function that aims at finding load balanced solutions. The objective function can be adapted at will depending upon the desired provider optimization objective or policies (such as load balancing to avoid congestion, consolidation, maximize revenue, etc.). We limit the scope of this chapter to a load balancing criterion in the optimization. In Table 5.1 we summarize the parameters and the variables used in the ILP formulation.

Objective function: A pragmatic approach to achieve load balancing is to select in priority candidate hosts that have the highest amount of free resources to gradually load the

TABLE 5.1: Table of Notations

Notation	**Description**
CPU_k	Residual capacity in a physical node k
P_{k_i,k_j}	Physical path interconnecting physical nodes k_i, 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, E_v	Set of virtual nodes, Set of 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}

infrastructure and distribute the load across nodes and links. This is accomplished using an objective function Z, defined in Equation (5.4.1), and combined with a set of equalities and inequalities reflecting the tenant constraints and the providers' obligations and interests. The variable min_{CPU}, in Equation (5.4.1), the smallest amount of available compute resources in all nodes in set C, constant over each run, serves as a normalization factor for the objective function. To maximize Z, the ILP selects in priority hosting nodes with the largest amount of available resources to ensure load balancing across the entire infrastructure.

$$Z = \sum_{i \in N_v} cpu_i \times \left(\sum_{k \in \mathcal{C}} \frac{CPU_k}{min_{CPU}} \times x_{ik}\right) \tag{5.4.1}$$

where the used binary variables are defined as follows:

$$x_{ik} = \begin{cases} 1, & \text{if the VNF } i \text{ is hosted on the substrate node } k; \\ 0, & \text{otherwise.} \end{cases} \tag{5.4.2}$$

$$y_{e_{ij},P_{k_i,k_j}} = \begin{cases} 1, & \text{if the virtual link } e_{ij} \text{ is mapped} \\ & \text{to physical path } P_{k_i,k_j}; \\ 0, & \text{otherwise.} \end{cases} \tag{5.4.3}$$

Furthermore, the following constraints must be satisfied in order to guarantee a feasible solution:

Node mapping constraint: ensures that each VNF i is mapped to exactly one physical candidate node k and its constituent components (VNFCs) are co-located in the same node.

$$\sum_{k \in \mathcal{C}} x_{ik} = 1, \;\; \forall i \in N_v \tag{5.4.4}$$

CPU capacity constraint: ensures that the available resources in a physical node k are sufficient to host VNF i.

$$\sum_{i \in N_v} cpu_i \times x_{ik} \leq CPU_k, \;\; \forall k \in \mathcal{C} \tag{5.4.5}$$

Link mapping constraint: makes sure that each 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. We compute $\mathcal{K}$-shortest paths candidates P_{k_i,k_j} using Dijkstra's algorithm based on residual bandwidth of links (not based on number of hops). It may find a longer but lightly loaded path better than the heavily loaded shortest path.

$$\sum_{k_i \in \mathcal{C}} \sum_{k_j \in \mathcal{C}} y_{e_{ij},P_{k_i,k_j}} = 1, \;\; \forall e_{ij} \in E_v, \, \forall i,j \in N_v \tag{5.4.6}$$

Bandwidth constraint: The residual bandwidth in a candidate physical path P_{k_i,k_j} has to satisfy the bandwidth requirement of virtual link e_{ij} to host the said link. We should not violate the remaining bandwidth of each physical link $e \in E_p$ on the physical path P_{k_i,k_j}.

$$\sum_{e_{ij} \in E_v} bw_{e_{ij}} \times y_{e_{ij},P_{k_i,k_j}} \times \delta_{ep} \leq BW_e, \quad \forall e \in P_{k_i,k_j},\ \forall P_{k_i,k_j} \in \mathcal{P} \tag{5.4.7}$$

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} having k_i as one of its endpoint or extremity (source). The other endpoint is k_j.

$$\sum_{k_j \in \mathcal{C}} y_{e_{ij},P_{k_i,k_j}} = x_{ik_i}, \quad \forall e_{ij} \in E_v,\ \forall k_i \in \mathcal{C} \tag{5.4.8}$$

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).

$$\sum_{k_i \in \mathcal{C}} y_{e_{ij},P_{k_i,k_j}} = x_{jk_j}, \quad \forall e_{ij} \in E_v,\ \forall k_j \in \mathcal{C} \tag{5.4.9}$$

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 each VNF i and j are mapped into nodes k_i and k_j with $k_i \neq k_j$.

$$x_{ik} + x_{jk} \leq 1, \quad \forall i,j \in N_v,\ \forall k \in \mathcal{C} \tag{5.4.10}$$

5.4.2 ILP with Reduced Number of Candidate hosts (R_ILP)

In addition to the ILP formulation of Equation (5.4.1), known not to scale since the VNF-FG embedding problem is NP-Hard, we propose a modified ILP that uses a reduced set of candidate hosts from the set of identified candidates, hosting nodes that meet the conditions and constraints in Equation (5.4.1). To accomplish this reduction, we select among the set of candidates a subset of much fewer candidates (typically 10 to 20 candidates are sufficient to obtain good solutions, i.e., sufficiently close from optimal). Among these subsets, we apply the ILP of Equation (5.4.1) to this subset to make the final mapping of the VNF-FG onto the subset. As mentioned earlier, the objective is to achieve load balancing to maximize the acceptance rate of requests and, thus, improve providers gains. The pseudo-code of the candidate subset selection is illustrated in Algorithm 4, where we select eligible hosts with more available resources (the $queue$ is sorted by available resources) to favor load balancing.

Algorithm 4: Candidate selection module pseudo-code

```
Inputs: $G_p$, $G_v$, $maxCand$
Output: A set of candidate hosts $\mathcal{C}$
$queue \leftarrow \emptyset$
$\mathcal{C} \leftarrow \emptyset$
foreach physical node $k \in N_p$ do
    $queue \leftarrow queue \cup k$
$stack \leftarrow$ Sort($queue$)
repeat
    $\mathcal{C} \leftarrow \mathcal{C} \cup$ Pop($stack$)
until size($\mathcal{C}$) = $maxCand$;
return $\mathcal{C}$
```

5.4.3 Batch placement and chaining algorithm (BR_ILP)

The current state of the art mainly focuses on online VNF-FG request embedding. Our objective is also to explore batch embedding and to propose a seminal work and analysis for this alternative approach to the VNF-FG placement and chaining problem. With the batch mode more client requests are likely to be accepted and served compared to the online mode. The number of accesses to the NFV-I in the batch mode is smaller. Mapping a batch of n requests requires only one access to the NFV-I while n accesses will be needed for the online mode. Operating in batch mode, by handling multiple requests over a viable time interval, can potentially provide benefits in terms of quality of the optimization with a likelihood of coming closer to optimal solutions. When operating online, the solutions can be instantaneously optimal but are typically suboptimal over longer time durations since there is no look-ahead nor combined look at past, present and future requests. The online solutions have the advantage of not incurring additional delays in providing embedding solutions on a sequential basis, processing one request at a time. Operating in batch mode enables, for instance, the processing of multiple requests as a group or an equivalent composite request. The batch mode will lead to solutions closer to optimal for the group of requests assuming the size of the batch and the interval over which the group is composed are adequately selected. Providers can reduce the cost of placing and chaining the multiple requests and hence improve their revenues as long as service level agreements with the tenants are met. The batch mode should lead to a better utilization of the provider physical infrastructure and the provider provisioned VNFs. The providers would logically have more freedom to improve or increase their profit and at the same time minimize the rejection rate of tenant requests. This can be beneficial to all stakeholders but we need to verify the conditions for such improvement compared to the online mode. In addition, the improvement must be significant to justify the implementation of the batch mode. The goal of our analysis and study is to set the stage for deeper investigations into the batch mode and facilitate the development of formal performance assessment models in the future.

At this stage, we conduct an initial study based on the batch embedding mode described in Figure 5.3 where there is a time window, denoted by W_b, during which the incoming requests are stored in a dedicated list denoted by $A(W_b)$. Arrivals and departures occur during such windows as depicted in the second batch window example in Figure 5.3. At the end of each batch window W_b there is a required processing period during which the provider executes a management policy to decide for instance i) the embedding order of the already arrived VNF-FG requests (the stored requests in $A(W_b)$) and ii) the requests to be accepted in case there are insufficient physical resources to host all the requests in the batch. This should always be done to achieve provider benefits and maximizing the satisfaction of tenants by keeping rejection rates to a strict minimum, i.e., to the smallest possible value depending on available resources during the batch. Our proposed batch-oriented algorithm applies the R_ILP to the batch and takes into account the scalability of our online R_ILP algorithm by limiting the batch size (retaining only the best candidate hosts) to obtain good solutions in reasonable times. The algorithm handles the queued VNF-FG requests for batch processing in a heuristic way. Instead of trying to place optimally the entire batch

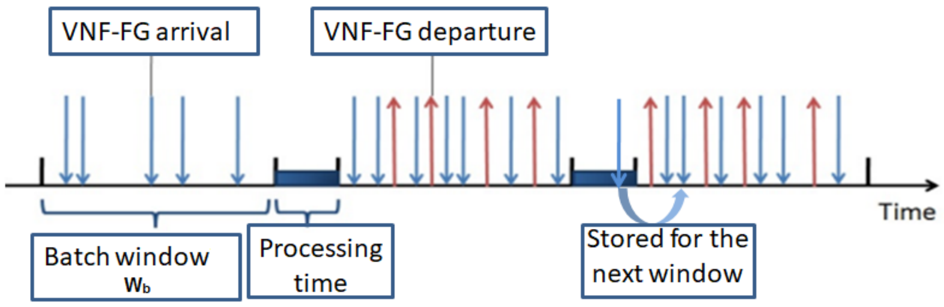

FIGURE 5.3: Processing of VNF-FGs in a batch mode

$A(W_b)$ (i.e., the stored incoming VNF-FGs during the batch window W_b), and end up rejecting all queued VNF-FG requests when there are no optimal placement solutions for the entire batch, the algorithm processes the VNF-FG requests by analyzing the whole batch to determine the order in which the requests must be treated to meet a selected objective such as maximizing provider gain, minimizing energy consumption, maximizing resource utilization, etc. In our performance assessment of the proposed batch-based algorithm, we sort the VNF-FG requests in the batch queue $A(W_b)$ according to the gains generated (foreseen) by their successful placement. The acceptance gain $\mathbb{G}(\mathcal{R}eq)$, of a given request $\mathcal{R}eq$, is formally expressed as defined in [112] and [113]:

$$\mathbb{G}(\mathcal{R}eq) = \sum_{i \in N_v} (cpu_i \times U_{cpu}) + \sum_{e_{ij} \in E_v} \left(bw_{e_{ij}} \times U_{bw}\right) \tag{5.4.11}$$

where U_{cpu} is the realized gain from allocating one unit of CPU resources and U_{bw} is the earned profit from the allocation of one bandwidth unit.

Requests generating the largest gains are placed (served) first in order to maximize overall achieved profit and at the same time avoiding the rejection of all the requests at once when they are treated jointly in one shot. This placement strategy limits rejection rate while simultaneously aiming at maximizing the provider benefits. As long as the rejection rate is kept low, below 5% for example, user satisfaction and quality of experience should remain acceptable.

The batch algorithm, Algorithm 5, operates as follows: upon arrival each incoming request Req_i is stored in the batch list $A(W_b)$. At the expiration of the batch window W_b, the queued VNF-FG requests are sorted based on their (expected) gain and stored in a new list $S(W_b)$.

Algorithm 5: Batch algorithm

```
1  Inputs: NFV-I, Batch window $W_b$
2  Output: Mapping of the incoming requests during $W_b$
3  $A(W_b) \leftarrow \emptyset$
4  while ($W_b$ not expired) do
5      if (arrival of new request $Req_i$) then
6          $A(W_b) \leftarrow A(W_b) \cup \{Req_i\}$
7  $S(W_b) \leftarrow sort(A(W_b))$
8  $M \leftarrow \emptyset$
9  while ($S(W_b)$ not empty) do
10     $\mathcal{R}eq \leftarrow extractTop(S(W_b))$
11     $m \leftarrow$ R_ILP(NFV-I, $\mathcal{R}eq$)
12     if ($m \neq null$) then
13         $M \leftarrow M \cup \{m\}$
14 retrun $M$
```

The ordered VNF-FG requests in $S(W_b)$ are processed sequentially by the online R_ILP algorithm. Successfully placed VNF-FG requests are stored in a list M along with their placement solution. Requests that cannot be placed are rejected. The placement process continues until all requests have been processed irrespective of the placement outcome in order to place as many requests as possible and, thus, reduce rejection rate. This is the simplest batch mode algorithm, called Batch R_ILP (BR_ILP) in our case.

5.4.4 EQL: Enhanced Q-Learning algorithm for VNF-FG Embedding

We propose an enhanced Q-Learning-based approach (EQL) for VNF-FG placement and chaining to improve long term performance and reward using reinforcement learning combined with an expert knowledge system. The other algorithms provide optimal solutions only for a given instant or over a rather short horizon and are actually quite suboptimal for longer term reward.

5.4.4.1 Preliminaries: Markov Decision Processes (MDP) and Reinforcement Learning (RL)

Markov Decision Processes (MDP) [122] give us a way to formalize sequential decision-making. The most important characteristic of an MDP is that the next state of the system only depends on the current state information and is unrelated to the earlier state. Unlike the Markov Process or Markov Chains, the MDP considers actions, that is, the next state of the system is related not only to the current state, but also to the current action taken. This mathematical formalization is the basis for structuring problems that are solved with Reinforcement Learning (RL) [123]. In fact, an MDP is used to describe an environment for RL, where the environment is fully observable.

The purpose of RL is to solve an MDP when the potentially visited states are unknown. One way to solve such a problem is to first learn the MDP and then solve it using algorithms such as value-iteration or policy-iteration (both using the Bellman equation [124]). The MDP can be learned by simulating different actions from each state until you have a high degree of confidence in the learned transition function and the learned reward function. Unfortunately, this is frequently not possible because the MDP is too large or it would be too expensive to learn the MDP.

Reinforcement Learning algorithms try to do both things at the same time: learn the MDP and solve it to find the optimal policy. To do that the algorithm needs to solve the exploration/exploitation tradeoff. Exploration means trying random actions (this helps discover the underlying MDP). Exploitation means following the so-far optimal policy (this helps maximize rewards). So RL is a technique to learn an MDP and solve it for the optimal policy at the same time. There are many algorithms related to reinforcement learning, such as Q-learning, State Action Reward State Action (SARSA), Deep Q-Network, Generative Adversarial Networks, etc. It has also been found that Q-Learning [125] is the most suitable for the problem studied in this chapter.

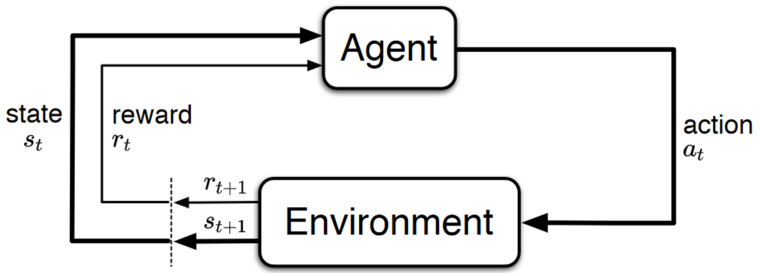

FIGURE 5.4: Operation algorithm of Q-Learning

As shown in Figure 5.4, in an MDP, we have a decision-maker, called an agent, that interacts with the environment it is placed in. These interactions occur sequentially over time. At each time step, the agent will get some representation of the environment state. Given this representation, the agent selects an action to take. The environment is then transitioned into a new state, and the agent is given a reward as a consequence of the previous action. This process of selecting an action from a given state, transitioning to a new state, and receiving a reward happens sequentially over and over again. Throughout this process, the agent's goal is to maximize the total amount of rewards that it receives from taking actions in given states. This means that the agent wants to maximize not just the immediate reward, but the cumulative rewards it receives over time.

5.4.4.2 Enhanced Q-Learning algorithm (EQL)

We optimize the Q-learning algorithm for the VNF-FG embedding problem. In this way, we can not only take advantage of the RL, but also avoid its defects, which can provide new ideas for dynamic SFC deployment. The Q-learning algorithm, shown in Equation (5.4.12), has a function that calculates the quality of a state-action combination:

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\bigg(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \bigg)}^{\text{learned value}} \tag{5.4.12}$$

In Equation (5.4.12), Q is a matrix that stores the recommended values of the executable actions in the current state. Depending on these values, the agent can decide which action to take next. In the Q matrix, t refers to the unit of time prior, $t + 1$: a unit of time post, max refers to the maximum value, s_t refers to the state space, which includes the resource requirements of VNFs and the remaining resources of the physical nodes, a_t to the action, s_{t+1} to the future state, and r_t is the reward value, which comes from the reward matrix R (r_t is the reward received when moving from the state s_t to the state s_{t+1}). The Q matrix is updated with reward r_t.

The core of the algorithm is a simple value iteration update, using the weighted average of the old value and the new information. The agent will repeat the above behavior until the Q matrix converges. Here, α and γ are the studied ratio between 0 and 1. The learning rate α determines to what extent newly acquired information overrides old information. A factor of 0 makes the agent learn nothing (exclusively exploiting prior knowledge), while a factor of 1 makes the agent consider only the most recent information (ignoring prior knowledge to explore possibilities). When the problem is stochastic, the algorithm converges under some technical conditions on the learning rate that require it to decrease to zero. In this chapter, a constant learning rate α is fixed to 0.1. The discount factor γ determines the importance of future rewards. A factor of 0 will make the agent short-sighted by only considering current rewards, while a factor approaching 1 will make it strive for a high long-term reward. In our implementation, we use a discount factor γ of 0.9.

In the reward matrix R, we set the value of an element equal to the residual capacity in each physical node (increasing or decreasing the reward r_t value according to the conditions, based on CPU, of each physical node). Through this, the load of nodes (in which each VNF operates) will be smartly distributed, so that each substrate node could exhibit the optimal performance through the efficient use of resources.

The proposed EQL algorithm operates in 5 steps:

1. compute the Q matrix using Equation (5.4.12) and learn the rules during the training process;

2. find a set of physical candidate nodes $\mathcal{C}$ related to a state s which are constituted by the method $getPolicyFromState$: that pick to move to the state that has the maximum Q value based on all possible actions; and the method $getExpertKnowledgeCandidates$: used to select in priority the physical nodes with highest available residual capacity to favor load balancing. This method accelerates the training process time, and also, avoids the random solutions at the beginning of the training process. Note that the node k available capacity and type constraints must be respected to be a candidate;

3. compute the shortest path between s and candidate k using Dijkstra's algorithm based on available bandwidth;

4. check link mapping: if links capacities are respected, confirm candidate k as a mapping solution for the VNF and add the solution path from s to k to the SolutionMapping $SolMap$ built so far;

5. update capacities on G_p, update R matrix (that will impact and update the Q matrix based on network conditions);

The pseudo-code of the EQL is shown in Algorithm 6.

5.5 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 request and hosting infrastructure 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, R_ILP, BR_ILP algorithms, and the enhanced Q-Learning algorithm (EQL). The performance of our proposed algorithms are also compared with the Eigendecomposition state of the art algorithm reported in [107] and a greedy solution [58] based on bipartite matching.

Algorithm 6: EQL algorithm pseudo-code

```
1  Inputs: $G_p$, $G_v$, $maxTrainCycles$, $maxCand$
2  Output: SolutionMapping $SolMap$
3  initialize the $Q$ matrix with all zero elements
4  initialize the $R$ matrix with the residual capacity in each physical node
5  $queue \leftarrow \emptyset$
6  A set of candidate hosts $\mathcal{C} \leftarrow \emptyset$
7  for $n = 0$ to $maxTrainCycles$ do
8      Compute $Q$ matrix using Equation (5.4.12)
9      if the Q matrix has basically converged then
10         Break; //return the Q matrix that can be used
11 foreach virtual link $e_{ij} \in E_v$ do
12     $s \leftarrow$ SelectInitialState(to host $vnf_i$ or $vnf_j$)
13     repeat
14         $\mathcal{C} \leftarrow \mathcal{C} \cup$ getPolicyFromState($s$) $\cup$ getExpertKnowledgeCandidates($s$)
15     until size($\mathcal{C}$) = $maxCand$;
16     $stack \leftarrow$ Sort($\mathcal{C}$)
17     while $stack \neq \emptyset$ and $solution$ = False do
18         $k \leftarrow$ Pop($stack$)
19         if ComputeLinkMapping($s$, $k$) = True and CheckLinks($s$, $k$) = True then
20             $solution \leftarrow$ True
21             $SolMap \leftarrow SolMap$ + GetPathSolution($s$, $k$)
22             Update capacities on $G_p$
23             Update R matrix
24 return $SolMap$
```

5.5.1 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 λ of 5 requests per 100 time units. The lifetime of each request follows an exponential distribution with a mean of 1000 time units. The Germany50 network topology [85] is used for the first assessment (with 50 nodes). The capacity of physical nodes and links are generated randomly in the $[100, 150]$ interval. The size of the VNF-FG requests is arbitrarily set to 5 nodes. The GT-ITM [86] tool is used to generate the requested VNF-FG topologies. The VNF-FG computing and bandwidth requirements are set to 10 resource units. 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 NFV-I sizes in the range $[100, 500]$ nodes and a connectivity of 0.3 (or 30%). The physical resource capacities (i.e., CPU and bandwidth) are also drawn randomly in the $[100, 150]$ 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 1000 time units. The size of the VNF-FG requests is also set to 5 nodes. The required CPU for each VNF in the VNF-FG is set to 10 units. The required bandwidth between two VNFs, to ensure communication between them, is set also to 10 units. 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. The ILP solver used in our experiments is Cplex [111].

5.5.2 Performance Metrics

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

- **Acceptance ratio** is the ratio of incoming service requests that have been successfully deployed on the network to all incoming requests. Maximizing this quantity is equivalent to minimizing rejection rate of the requests.
- **Acceptance gain** is the service provider realized profit at time t when accepting client requests (in our case successful placement of VNF-FG requests). The acceptance gain is formally expressed as:

$$\mathbb{G}(t) = \sum_{\mathcal{R}eq \in \mathcal{AR}_t} \mathbb{G}\left(\mathcal{R}eq\right) \tag{5.5.13}$$

  where $\mathcal{AR}_t$ is the set of accepted requests up to time t and $\mathbb{G}(\mathcal{R}eq)$ is the request $\mathcal{R}eq$ gain as defined in Eq. (5.4.11). Note that $\mathbb{G}$ is the total profit taking into consideration all the incoming requests.

- **Execution time** is the time needed to find an embedding solution for one VNF-FG request. This metric reflects the ability of the algorithms to scale with problem size and this is especially important for the ILP algorithm assessment.

5.5.3 Simulation Results

The performance of the algorithms is reported for the Germany50 network topology for small scale problems (50 nodes) and for randomly generated network topologies for larger problems involving hundreds of nodes and links.

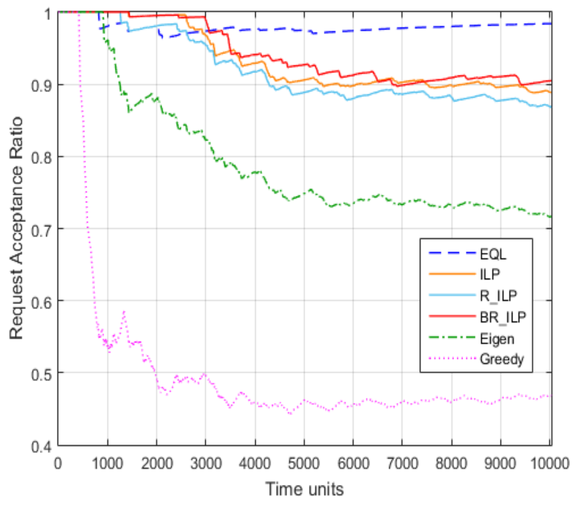

FIGURE 5.5: Acceptance ratio

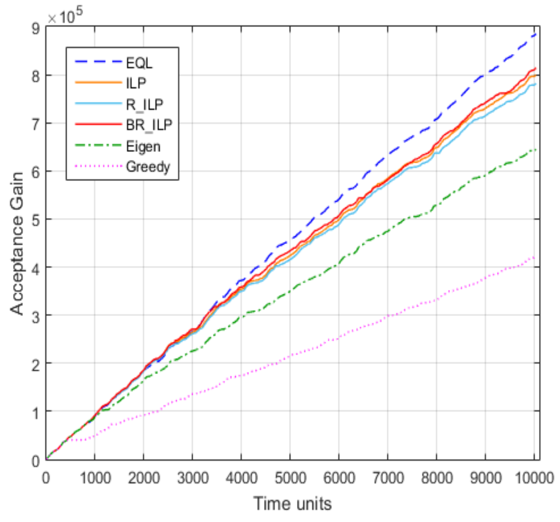

FIGURE 5.6: Acceptance gain

5.5.3.1 Realistic topology evaluation

For the Germany50 network, since the ILP can provide placement solutions in reasonable execution times it can be included in the performance comparisons with the R_ILP, BR_ILP, EQL, Eigen [107], and Greedy [58]. The results provide insight on these five algorithms compared to the optimal solutions provided by the ILP for such small problem sizes. The performance is pushed further for much larger networks for the heuristic algorithms only since they scale much better with problem size. As mentioned, the larger graphs are randomly generated with the GT-ITM Tool.

Figure 5.5 depicts the results in terms of acceptance ratio. The EQL algorithm outperforms the ILP based solutions (i.e., ILP, R_ILP, BR_ILP) by accepting more requests in long-term. The normalized metric (acceptance ratio) confirms these results passed a transient state (passed 3000 time units) where the achieved ratios are around 98.4% for the EQL, 90.5% for the BR_ILP, 88.9% for the ILP, 86.9% for the R_ILP, 71.7% for the Eigendecomposition-based algorithm, and only 46.8% for the Greedy algorithm. The Eigendecomposition approach is known to provide much better results for highly connected topology compared with weakly connected graphs.

The computed acceptance gain depicted in Figure 5.6, that corresponds to the total generated profit at time t, confirm the relative performance of the algorithms observed for the acceptance ratio. The EQL achieved provider gain is 8.02% higher than the BR_ILP, and respectively 9.65%, 11.68%, 27.13%, and 52.43% better than ILP, R_ILP, Eigen, and Greedy.

Figures 5.7 compares the quality realized by the proposed algorithms in terms of objective function, achieved maximum in Equation (5.4.1), to provide insight on the relative performance of the heuristics to the optimal solutions found by the ILP. Since we aim in this chapter at maximizing provider profit, 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. Figure 5.7 reports the achieved Z values and indicates clearly that the closest in performance to the ILP is the enhanced Q-Learning algorithm (EQL). The batch ILP algorithm (BR_ILP) is outperformed by the EQL in terms of quality of the solutions especially for the long-term evaluation.

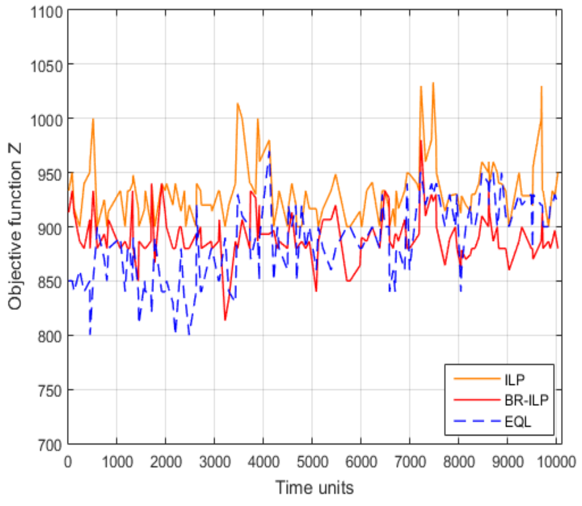

FIGURE 5.7: Quality of the mapping

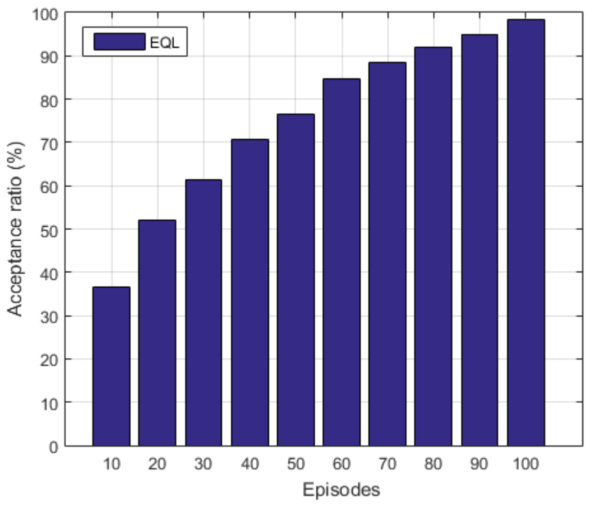

FIGURE 5.8: Percentage of deployed VNF-FGs

Figure 5.8 reports the average acceptance percentage of the enhanced Q-Learning (EQL) algorithm as a function of increasing the number of episodes (from 10 to 100). We observe that the average acceptance percentage under a high number of episodes exhibits much better performance than under a low number of episodes (98.4% for 100 episodes versus 36.5% for 10 episodes).

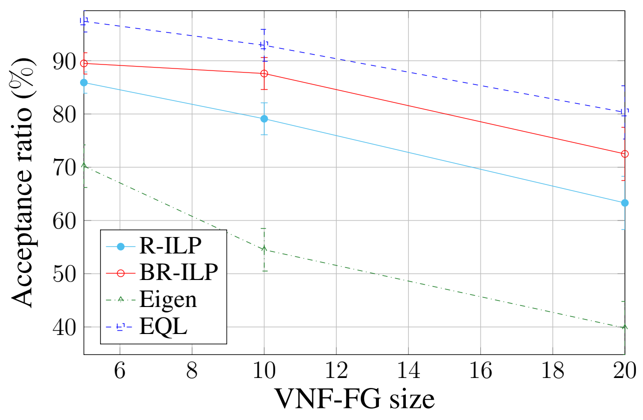

FIGURE 5.9: Acceptance percentage w.r.t VNF-FG size variation

5.5.3.2 Large-scale evaluation

To analyze the behavior of the algorithms for larger problem sizes, NFV-Is with 200 nodes are generated and used in this second performance assessment. Figure 5.9 reports the average acceptance percentage of the algorithms as a function of increasing VNF-FG request sizes (from 5 to 20) and confirms the previous findings. The EQL algorithm outperforms the ILP based solutions (i.e., R_ILP, BR_ILP) and the Eigen approach by accepting more requests. The results in Figure 5.9, presented with a 95% confidence interval, depict a high similarity with results in Figure 5.5 (more precisely for VNF-FG size of 5 nodes).

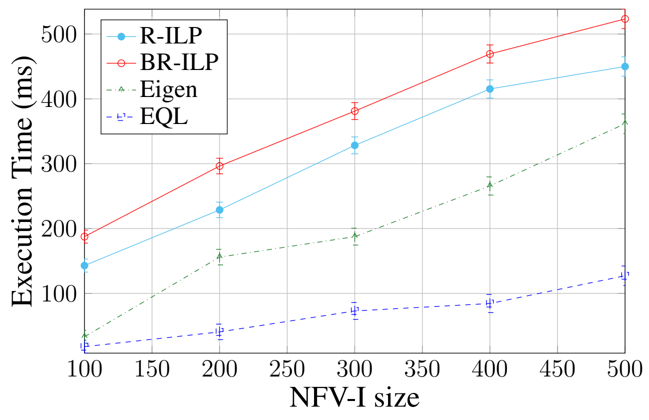

FIGURE 5.10: Execution Time w.r.t NFV-I size variation

5.5.3.3 Execution time (Convergence time)

Figure 5.10 reports the execution time performance of the algorithms with problem size to gain insight on their scalability and complexity. We generate 1000 VNF-FG requests and vary the substrate graph sizes from 100 to 500 nodes. The EQL based algorithm has significantly better execution time compared with the R_ILP, BR_ILP, and Eigen execution times.

5.6 Conclusion

In this chapter, we propose approaches for the VNF-FG placement and chaining problem in an online and a batch mode. The online strategy exploits an Integer Linear Program derived from a mathematical model of the VNF-FG embedding problem. The ILP model is derived and used to find optimal placement solutions to host the requests when problem size is small. To control complexity and improve scalability for larger graph sizes, the ILP explores only a reduced subset of candidate hosts (R_ILP). The introduced batch mode algorithm, to enhance further overall performance, uses our R_ILP algorithm to process a group of requests, queued in a batch window, by serving in priority requests that generate higher profit for the providers while aiming at maximizing the number of served (i.e., placed, chained, hosted) requests. An Enhanced Q-Learning-Based Approach (EQL) is also proposed and compared with the ILP solutions. The EQL algorithm improves the efficiency of addressing this specific type of problem (it not only capitalizes on the decision-making advantages of RL, but also avoids a lengthy training process by injecting expert knowledge). The experimental results show that the performance of the proposed EQL algorithm is superior to that of the ILP solutions, the greedy, and the eigendecomposition algorithms when processing service requests in long-term. In fact, its performance advantages are reflected by the decision time, quality of mapping, deployment success rate and deployment profit when deploying SFCs.