An Enhanced Reinforcement Learning Approach for Dynamic Placement of Virtual Network Functions

Omar Houidi*, Oussama Soualah*, Wajdi Louati†, and Djamal Zeghlache*
*Telecom SudParis, Samovar-UMR 5157 CNRS, Institut Polytechnique de Paris, France
†ReDCAD Lab, University of Sfax, Tunisia
Email: {omar.houidi, oussama.soualah, djamal.zeghlache}@telecom-sudparis.eu,
wajdi.louati@redcad.org

***Abstract*—This paper addresses Virtualized Network Function Forwarding Graph (VNF-FG) embedding with the objective of realizing long term reward compared to placement algorithms that aim at instantaneous optimal placement. The long term reward is obtained using Reinforcement Learning (RL), following a Markov Decision Process (MDP) model, enhanced through the injection of expert knowledge in the learning process. A comparison with an Integer Linear Programming (ILP) approach, a reduced candidate set (R-ILP), and an algorithm that treats the requests in batch reveals the potential improvements using the RL approach. The instantaneous and short term reward solutions are efficient only in finding instant solutions as they make decisions only on current infrastructure status for a given request at a time or eventually a batch of requests. They are efficient only for present conditions without anticipating future requests. RL possesses instead the learning and anticipation capabilities lacking in instantaneous and snapshot optimizations. A Reinforcement Learning based approach, called EQL (Enhanced Q-Learning), aiming at balancing the load on hosting infrastructures is proposed to achieve the desired longer term reward. EQL employs RL to learn the network and control it based on the usage patterns of the physical resources. Results from extensive simulations, based on realistic and large scale topologies, report the superior performance of EQL in terms of acceptance rate, quality, scalability and achieved gains.**

***Index Terms*—Network Function Virtualization, Reinforcement Learning, Dynamic Service Placement, Optimization.**

I. INTRODUCTION

Network Function Virtualization (NFV) [1] decouples network functions (such as firewalls, load balancers, etc.) from dedicated hardware devices (the traditional middleboxes). This decoupling enables hosting of network services, known as Virtualized Network Functions (VNFs), on commodity hardware (such as servers), facilitates and accelerates service deployment and management by providers, improves flexibility, leads to efficient and scalable resource usage, and reduces costs.

This paradigm is a major evolution for networking with nevertheless new challenges to address especially the placement and chaining of VNFs in hosting infrastructures to support multiple tenants and applications. This paper focuses on the optimal placement of networking functions and services and their chaining that meet tenant requests and lead to efficient use of hosting infrastructure resources. The objective is to propose placement algorithms that minimize the percentage of rejected tenant requests. This paper addresses more specifically 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 but mostly to achieve instantaneous and sequential placement of the requests resulting 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). A practical way to resolve the longer term reward placement problem is to use Reinforcement Learning (RL) as opposed to aiming exclusively at instantaneous optimality using traditional combinatorial and convex optimization.

To highlight the benefits of reinforcement learning applied to the VNF-FG and SFC placement problem, we compare the performance to an Integer Linear Program (ILP) using an exact formulation for instantaneous optimal placement handling one request at a time. All algorithms aim at balancing the load throughout the hosting infrastructure. Other criteria than load balancing can be selected, such as 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. 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 but this is accomplished at the expense of optimality. A batch algorithm, called BR-ILP, that operates on a group of requests, handled as a composite graph request, is included in the comparison since it is expected to perform better than the ILP in terms of placement efficiency over longer time scales. The RL solution uses standard Q-Learning enhanced with expert knowledge to accelerate learning during training and when unknown conditions and situations emerge.

Section II presents related work. The problem formulation is described in Section III. The proposals are introduced in Section IV. Section V reports the performance evaluation results and Section VI summarizes the main findings.

978-1-7281-4490-0/20/$31.00 © 2020 IEEE

II. Related work

The need to dynamically deploy virtualized network services on-demand, through VNF-FG embedding, is identified as a core technology of 5G networks. Therefore, this issue has been at the very center of academic and industrial research in recent years. As comprehensive surveys are already given in [1] and [2], we only give a short summary of VNF-FG embedding related works, as well as the use of RL in networking and especially in the VNF-FG placement and chaining domain.

VNF-FG embedding is formulated as an ILP in [3] and [4] to find exact solutions for hosting the VNFs of the requested service graph. As the addressed problem is NP-Hard, the exact solutions do not scale with the size and require an excessive amount of time to find the optimal solutions.

Since 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 propose RL-based approaches for VNF-FG embedding.

Authors in [5] 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 [6], 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 [7], authors propose a deep reinforcement learning-based approach for multi-domain VNF-FG embedding. However, the results are only obtained on a single small network topology BtEurope of 24 nodes. Authors in [8] 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.

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 embedding. In fact, the major difference between our approach and the existing works is that the training phase in the EQL is accelerated by injecting additional driving knowledge.

III. 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.

A. 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

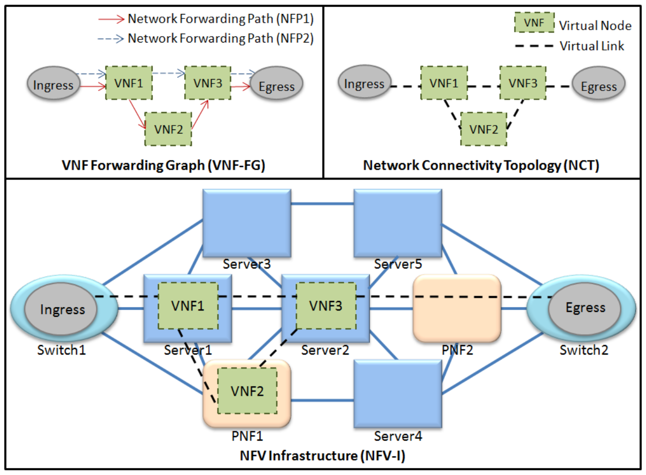


Fig. 1: Mapping of the VNF-FG in the NFV-I.

Infrastructure (NFV-I) in [9], 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, 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, two switches, and some interconnected servers is presented in Fig. 1.

B. VNF Forwarding Graph or SFC Graph

The client request 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 $bw_{e_{ij}}$. 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. Fig. 1 depicts two Network Forwarding Paths (NFP) or chains. 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.

TABLE I: Main 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 $P_{k_i,k_j} \in \mathcal{P}$: $\delta_{ep} = 1 \Leftrightarrow e \in P_{k_i,k_j}$
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}

C. VNF-FG placement and chaining

Fig. 1 depicts a placement solution for the 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, improving energy efficiency, maximizing acceptance ratios [10], and improving provider gains [11].

IV. 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.

A. 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. In Table I 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 infrastructure and distribute the load across nodes and links.

This is accomplished using an objective function $\mathbb{Z}$, defined in Eq. (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 Eq. (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 $\mathbb{Z}$, the ILP selects in priority hosting nodes with the largest amount of available resources to ensure load balancing across the entire infrastructure.

$$\begin{aligned}\mathbb{Z} = &\Big(\textstyle\sum_{i\in N_v} cpu_i \times (\sum_{k\in\mathcal{C}} \frac{CPU_k}{min_{CPU}} \times x_{ik})\Big)\\ &+ \Big(\textstyle\sum_{e_{ij}\in E_v} bw_{e_{ij}} \times (\sum_{k_i\in\mathcal{C}}\sum_{k_j\in\mathcal{C}} y_{e_{ij},P_{k_i,k_j}})\Big)\end{aligned} \tag{1}$$

where the used binary variables are defined as follows:

$$x_{ik} = \begin{cases} 1, & \text{if the VNF } i \text{ is hosted on the substrate node } k;\\ 0, & \text{otherwise.}\end{cases} \tag{2}$$

$$y_{e_{ij},P_{k_i,k_j}} = \begin{cases} 1, & \text{if the virtual link } e_{ij} \text{ is mapped}\\ & \text{to physical path } P_{k_i,k_j};\\ 0, & \text{otherwise.}\end{cases} \tag{3}$$

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.

$$\sum_{k\in\mathcal{C}} x_{ik} = 1, \ \ \forall i \in N_v \tag{4}$$

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

$$\sum_{i\in N_v} cpu_i \times x_{ik} \leq CPU_k, \ \ \forall k \in \mathcal{C} \tag{5}$$

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.

$$\sum_{k_i\in\mathcal{C}}\sum_{k_j\in\mathcal{C}} y_{e_{ij},P_{k_i,k_j}} = 1, \ \ \forall e_{ij} \in E_v, \ \forall i,j \in N_v \tag{6}$$

Bandwidth constraint: Obviously 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}.

$$\begin{aligned}\sum_{e_{ij}\in E_v} bw_{e_{ij}} \times y_{e_{ij},P_{k_i,k_j}} \times \delta_{ep} \leq BW_e,\\ \forall e \in P_{k_i,k_j}, \ \forall P_{k_i,k_j} \in \mathcal{P}\end{aligned} \tag{7}$$

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.

$$\sum_{k_j\in\mathcal{C}} y_{e_{ij},P_{k_i,k_j}} = x_{ik_i}, \ \ \forall e_{ij} \in E_v, \ \forall k_i \in \mathcal{C} \tag{8}$$

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).

$$\sum_{k_i \in \mathcal{C}} y_{e_{ij}, P_{k_i,k_j}} = x_{jk_j}, \ \forall e_{ij} \in E_v, \ \forall k_j \in \mathcal{C} \tag{9}$$

B. ILP with Reduced Number of Candidate hosts (R-ILP)

In addition to the ILP formulation of Eq. (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 Eq. (1). To accomplish this reduction, we select among the set of candidates a subset of much fewer candidates (eligible hosts with more available resources). We apply the ILP of Eq. (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 and, thus, improve providers gains.

C. 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. The batch mode should lead to a better utilization of the provider physical infrastructure and the provider provisioned VNFs.

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. Requests generating the largest gains are placed 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. Our batch algorithm, called Batch R-ILP (BR-ILP) is more detailed in [12].

D. EQL: Enhanced Q-Learning algorithm for VNF-FG Embedding

We propose an enhanced Q-Learning-based approach (EQL) for VNF-FG embedding to improve long term performance and reward using RL 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.

1) ***Preliminaries: Markov Decision Processes (MDP) and Reinforcement Learning (RL):*** Markov Decision Processes (MDP) [13] 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. 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) [14]. In fact, an MDP is used to describe an environment for RL, where the environment is fully observable.

There are many algorithms related to reinforcement learning, such as Q-learning, State Action Reward State Action (SARSA), Deep Q-Network, etc. After comparison, it has been found in [15] that Q-Learning [16] is the most suitable for the problem studied in this paper.

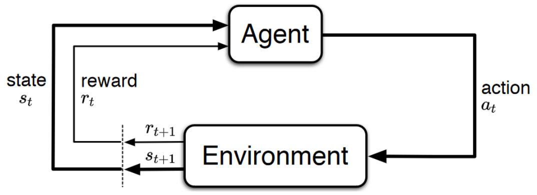


Fig. 2: Operation algorithm of Q-Learning.

As shown in Fig. 2, in an MDP, we have a decision-maker, called 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 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.

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. The Q-learning algorithm, shown in Eq. (10), has a function that calculates the quality of a state-action combination:

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \underbrace{\Big(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \Big)}_{\text{learned value}} \tag{10}$$

In Eq. (10), 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, 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. The discount factor γ determines the importance of future rewards. In our implementation, we use a learning rate α of 0.1 and 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 Eq. (10) 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 1.

Algorithm 1: 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 Eq. (10)
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 $SolMap$
```

V. Performance Evaluation

The algorithms are compared using extensive simulations where both the request and hosting infrastructure are drawn using standard graph generation tools and real infrastructure topologies. The comparison includes the ILP, R-ILP, BR-ILP algorithms, and the enhanced Q-Learning algorithm (EQL).

A. Simulation Environment

The simulations are performed on a virtual machine (with 2.50 GHz Quad vCore and 6 GBytes of available vRAM) instantiated in our experimental cloud and networking platform. Our proposed algorithms are integrated (as a VM) in the smart placement module, acting as one key component of the cloud framework, responsible for VNF-FGs embedding. 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 [17] 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 tool is used to generate the requested VNF-FG topologies. The required capacities of the VNF-FG are fixed to 10 CPU units for each virtual node and to 10 bandwidth units for each virtual link. The connectivity between nodes in the VNF-FG is set to 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 resources 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 is set also to 10 units. The connectivity between nodes in the VNF-FG is set to 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.

B. Performance Metrics

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

*1) **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.

*2) **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:

$$\mathbb{G}(t) = \sum_{\mathcal{R}eq \in \mathcal{AR}_t} \mathbb{G}\left(\mathcal{R}eq\right) \tag{11}$$

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 expressed as:

$$\mathbb{G}(\mathcal{R}eq) = \sum_{i \in N_v} \left(cpu_i \times U_{cpu}\right) + \sum_{e_{ij} \in E_v} \left(bw_{e_{ij}} \times U_{bw}\right) \tag{12}$$

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. Note that $\mathbb{G}$ is the total profit taking into consideration all the incoming requests.

*3) **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.

C. 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.

*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, and EQL. The results provide insight on these three 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.

Fig. 3(a) 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, and 86.9% for the R-ILP.

The computed acceptance gain depicted in Fig. 3(b), 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%, and 11.68% better than the ILP, and the R-ILP.

Fig. 3(c) compares the quality realized by the proposed algorithms in terms of objective function, achieved maximum in Eq. (1), to provide insight on the relative performance of the heuristics to the optimal solutions found by the ILP. Since we aim in this paper at maximizing provider profit, we define the mapping quality based on the value of the objective function $\mathbb{Z}$ that describes the algorithms ability to achieve load balancing on the hosting infrastructure. Fig. 3(c) reports the achieved $\mathbb{Z}$ 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.

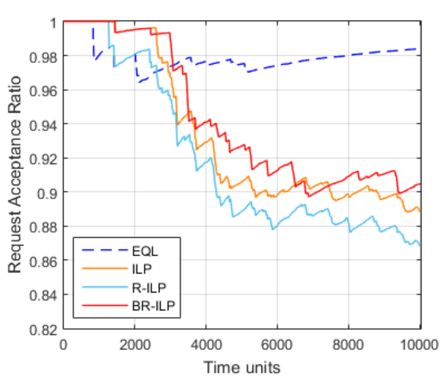
Fig. 4 reports the average acceptance percentage of the EQL algorithm as a function of increasing the number of episodes (from 10 to 100). We observe that the average acceptance percentage under a high number of episodes exhibits much better performance than under a low number of episodes (98.4% for 100 episodes versus 36.5% for 10 episodes).

*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. Fig. 5 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) by accepting more requests. The results in Fig. 5, presented with a 95% confidence interval, depict a high similarity with results in Fig. 3(a) (more precisely for VNF-FG size of 5 nodes).

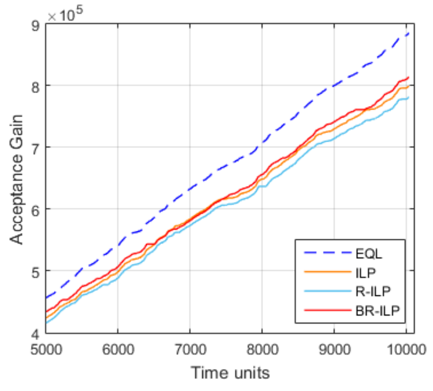
*3) **Execution time (Convergence time)**:* Fig. 6 reports the execution time performance of the algorithms with problem size to gain insight on their scalability and complexity. We generate 1000 VNF-FG requests and vary the substrate graph sizes from 100 to 500 nodes. The EQL based algorithm has significantly better execution time compared with the R-ILP and BR-ILP execution times.

VI. Conclusion

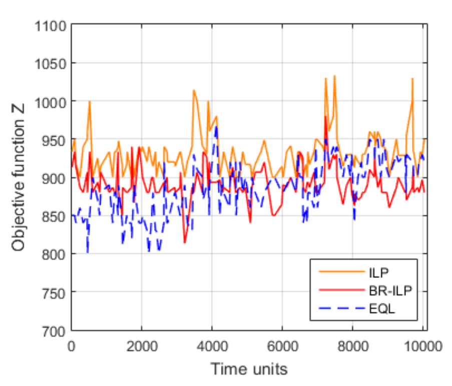
In this paper, we propose approaches for the VNF-FG embedding problem in an online and a batch mode. The online strategy exploits an ILP derived from a mathematical model of the VNF-FG embedding problem. The ILP model is derived and used to find optimal placement solutions to host the requests when problem 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 uses our R-ILP algorithm



(a) Acceptance ratio



(b) Acceptance gain



(c) Quality of the mapping

Fig. 3: Germany50 network topology results (NFV-I size = 50).

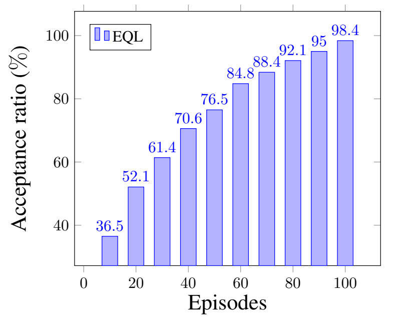


Fig. 4: Percentage of deployed VNF-FGs (NFV-I size = 50).

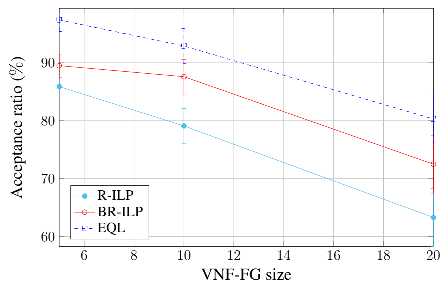


Fig. 5: Acceptance percentage w.r.t VNF-FG size variation (NFV-I size = 200, VNF-FG size = up to 20).

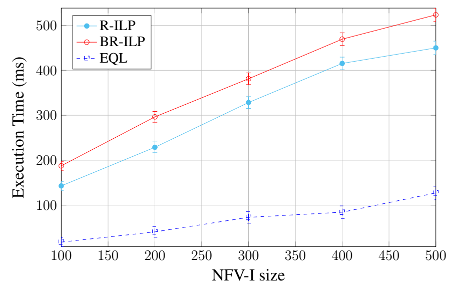


Fig. 6: Execution Time w.r.t NFV-I size variation (NFV-I size = up to 500, VNF-FG size = 5).

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 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 EQL algorithm is superior to that of the ILP solutions when processing service requests in long-term.

References

[1] B. Yi, X. Wang, K. Li, S. K. Das, and M. Huang, "A comprehensive survey of Network Function Virtualization," *Computer Networks*, vol. 133, pp. 212–262, 2018.

[2] J. Gil-Herrera and J. F. Botero, "Resource Allocation in NFV: A Comprehensive Survey," *IEEE Trans. Network and Service Management*, vol. 13, no. 3, pp. 518–532, 2016.

[3] H. Moens and F. D. Turck, "VNF-P: A model for efficient placement of virtualized network functions," in *CNSM 2014 and Workshop, Rio de Janeiro, Brazil, November 17-21, 2014*, 2014, pp. 418–423.

[4] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *IEEE CloudNet 2014, Luxembourg, Luxembourg, October 8-10, 2014*, 2014, pp. 7–13.

[5] S. Haeri and L. Trajkovic, "Virtual Network Embedding via Monte Carlo Tree Search," *IEEE Trans. Cybernetics*, vol. 48, no. 2, pp. 510–521, 2018.

[6] R. Mijumbi, J. Gorricho, J. Serrat, M. Claeys, F. D. Turck, and S. Latré, "Design and evaluation of learning algorithms for dynamic resource management in virtual networks," in *IEEE NOMS 2014, Krakow, Poland, May 5-9, 2014*, 2014, pp. 1–9.

[7] P. T. A. Quang, A. Bradai, K. D. Singh, and Y. H. Aoul, "Multi-domain non-cooperative VNF-FG embedding: A deep reinforcement learning approach," in *IEEE INFOCOM Workshops 2019, Paris, France, April 29 - May 2, 2019*, 2019, pp. 886–891.

[8] S. I. Kim and H. S. Kim, "A research on dynamic service function chaining based on reinforcement learning using resource usage," in *ICUFN 2017, Milan, Italy, July 4-7, 2017*, 2017, pp. 582–586.

[9] ETSI GS NFV 001: "Network Functions Virtualisation (NFV); Use Cases", 2013.

[10] O. Houidi, O. Soualah, W. Louati, and D. Zeghlache, "Dynamic VNF Forwarding Graph Extension Algorithms," *IEEE Transactions on Network and Service Management*, 2020.

[11] O. Houidi, O. Soualah, W. Louati, D. Zeghlache, and F. Kamoun, "Virtualized Network Services Extension Algorithms," in *17th IEEE NCA 2018, Cambridge, MA, USA, November 1-3, 2018*, 2018, pp. 1–5.

[12] O. Soualah, M. Mechtri, C. Ghribi, and D. Zeghlache, "Online and batch algorithms for VNFs placement and chaining," *Computer Networks*, vol. 158, pp. 98–113, 2019.

[13] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, ser. Wiley Series in Probability and Statistics. Wiley, 1994.

[14] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey," *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, 1996.

[15] J. Sun, G. Huang, G. Sun, H. Yu, A. K. Sangaiah, and V. Chang, "A Q-Learning-Based Approach for Deploying Dynamic Service Function Chains," *Symmetry*, vol. 10, no. 11, p. 646, 2018.

[16] C. J. C. H. Watkins and P. Dayan, "Technical Note Q-Learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[17] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski, "SNDlib 1.0–Survivable Network Design Library," *Networks*, vol. 55, no. 3, pp. 276–286, 2010.