Virtual Network Function Placement Optimization With Deep Reinforcement Learning

Ruben Solozabal, Josu Ceberio, Aitor Sanchoyerto, Luis Zabala, Bego Blanco, and Fidel Liberal

***Abstract*— Network Function Virtualization (NFV) introduces a new network architecture framework that evolves network functions, traditionally deployed over dedicated equipment, to software implementations that run on general-purpose hardware. One of the main challenges for deploying NFV is the optimal resource placement of demanded network services in the NFV infrastructure. The virtual network function placement and network embedding can be formulated as a mathematical optimization problem concerned with a set of feasibility constraints that express the restrictions of the network infrastructure and the services contracted. This problem has been reported to be NP-hard, as a result most of the optimization work carried out in the area has focused on designing heuristic and metaheuristic algorithms. Nevertheless, in highly constrained problems, as in this case, inferring a competitive heuristic can be a daunting task that requires expertise. Consequently, an interesting solution is the use of Reinforcement Learning to model an optimization policy. The work presented here extends the *Neural Combinatorial Optimization* theory by considering constraints in the definition of the problem. The resulting agent is able to learn placement decisions by exploring the NFV infrastructure with the aim of minimizing the overall power consumption. The experiments conducted demonstrate that when the proposed strategy is also combined with heuristics, highly competitive results are achieved using relatively simple algorithms.**

***Index Terms*— Constrained combinatorial optimization, Reinforcement Leaning, 5G, NFV.**

I. Introduction

CURRENT network deployments based on specific-purpose hardware have many drawbacks: they have high capital & operational expenditures, do not allow their functionality to be updated in a simple way, and have short life-cycles. As a result, Network Function Virtualization (NFV) has emerged from the industry, promising to transform the way that network operators design, manage and deploy their network infrastructure thanks to the current advances in virtualization technologies. In order to standardize this technology, in 2012, leading telecom network operators created a specification group in the European Telecom Standards Institute (ETSI). The resulting NFV architecture [1] replaces traditional service-specific hardware with software modules that handle specific Virtual Network Functions (VNF). This architecture provides the modularity and isolation for each function, so that they can operate independently inside a general purpose virtual environment.

In the NFV framework, a virtualized Network Service (NS) is created combining individual VNFs. The composition of the ordered set of VNFs to be deployed over the infrastructure is referred to as a Forward Graph. In order to fully define a network service in an NFV environment, it is necessary to specify: the VNF components that form the network service, their respective order inside the graph and its placement in the NFV infrastructure. One of the main challenges that this technology faces is the optimization of resource placement in the underlaying network infrastructure. This challenge is known as the VNF Forward Graph Embedding problem (VNF-FGE) and is one of the NFV Resource Allocation problems, together with the VNF Chaining Composition and VNF Scheduling problems.

In this paper, we focus on optimizing the VNF-FGE. This problem consists of efficiently mapping a set of network service requests on top of the physical network infrastructure. Particularly, we seek to obtain the optimal placement for a NS chain considering the state of the virtual environment, such that a specific resource objective is accomplished (e.g., maximization of the remaining resources, minimization of the overall power consumption, optimization of a specific QoS metric, etc.). Besides, specific aspects of NFV, such as forwarding latency, ingress/egress bitrate and flow chaining, have to be taken into account.

We formalize the VNF-FGE as a constrained combinatorial optimization problem, where NSs need to be placed on top of the network infrastructure meeting the Service Level Agreements. This problem has been expressed as NP-hard [2]; therefore, it is only possible to exactly solve it for small instances. Most of the existing approaches in the literature are focused on the design of heuristic or metaheuristic algorithms. Nevertheless, these strategies present a mayor drawback. Despite their good results in solving constrained optimization problems, when the number of restrictions is high and the feasible region of solutions is small, they tend, in some cases, to be ineffective.

Manuscript received June 23, 2019; revised October 17, 2019; accepted November 6, 2019. Date of publication December 30, 2019; date of current version February 19, 2020. This work was supported in part by the European Commission through the 5G-PPP Project ESSENCE of Horizon 2020 Research of the European Union under Grant 761592, in part by the Spanish Ministry of Economy, Industry and Competitiveness through the projects TIN2016-78365-R and 5RANVIR under Grant TEC2016-80090-C2-2-R, and in part by the Basque Government IT1003-16 and ELKARTEK Programs. *(Corresponding author: Ruben Solozabal.)*

R. Solozabal, A. Sanchoyerto, L. Zabala, and F. Liberal are with the Networking Quality and Security Department, University of the Basque Country (UPV/EHU), 48013 Bilbao, Spain (e-mail: ruben.solozabal@ehu.eus; aitor.sanchoyerto@ehu.eus; luis.zabala@ehu.eus; fidel.liberal@ehu.eus).

J. Ceberio is with the Department of Computer Science and Artificial Intelligence, University of the Basque Country, 20080 Donostia, Spain (e-mail: josu.ceberio@ehu.eus).

B. Blanco is with the Department of Computer Languages and System, University of the Basque Country (UPV/EHU), 48013 Bilbao, Spain (e-mail: begona.blanco@ehu.eus).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSAC.2019.2959183

Recently, a novel method to approach combinatorial optimization problems using Reinforcement Learning (RL) has emerged, Neural Combinatorial Optimization (NCO) (Bello et al. 2016) [3]. This approach has proven to be able to learn near optimal solutions on small classical combinatorial problems such as TSP, Bin Packing or VRP [4]–[6]. Specifically, NCO estimates a Neural Network (NN) model to describe the relation between the space of instances of the problem and the solutions for each of them. The parameters of the model are estimated iteratively by means of an RL approach. Particularly, it takes instances from the problem space, and uses the reward (cost value in our problem) obtained from evaluating the solution returned by the neural network to improve the accuracy of the model. The effectiveness that NCO has shown by modeling optimization policies has motivated us to use it on the naturally constrained VNF-FGE problem.

In the VNF-FGE, an instance of the problem is defined as a service request and the description of the infrastructure in the deployment moment. In addition to optimizing the cost function, in this problem there are a number of restrictions that need to be complied. In order to approach such types of problems with NCO, we extend that paradigm and conduct a reward policy optimization taking into account the constraints (Tessler et al. 2018) [7]. Particularly, we implement a sequence-to-sequence model that, for a given service chain request, and taking into account the state of the NFV infrastructure, infers a placement policy aiming to minimize the overall power consumption. As will be discussed, a major simplification has been undertaken to use this neural architecture. The networking has been reduced to a star connection to simplify the path selection in the problem.

For the sake of validating our proposal, we conduct an experimental study where we compare the feasibility and quality of the solutions obtained by the neural network model against the open-source constraint solver *Gecode* [8] and a First-Fit (FF) algorithm designed originally for the Multi-resource Bin Packing problem. During the experimentation, two scenarios are proposed: small and large infrastructures. This way we perform a comparison in environments that benefits the solver and the heuristic respectively. Results indicate that when the model is used in conjunction with the heuristic algorithm, the most competitive solutions are achieved for both scenarios. We will refer to this method as the Hybrid-Agent.

The remainder of the paper is organized as follows. In Section II, the background for the VNF-FGE literature is presented and the Neural Combinatorial Optimization theory is introduced. Next, the optimization problem is mathematically formalized in Section III. Afterwards, in Section IV, the proposed optimization approach and the architectural details are described. Section V contains the experimental study. Finally, the paper concludes in Section VI with a summary and some ideas for future research.

II. Background

A. Literature of VNF-FGE Problem

Several previous works have undertaken the VNF-FGE problem. In the literature, we can find it present in [9] or [10]. As it is an NP-hard optimization problem, run-times to optimally solve it are unaffordable for large instances. For this reason, this problem has been tackled by applying the following alternatives:

i) Optimal solutions can be achieved for small problem instances. For example, using Mixed Integer Linear Programming [11], [12]. Or analytically solving the Bellman equation on the problem described as a Markov Decision Process [13].

ii) Another alternative is the heuristic approaches that, despite not guaranteeing convergence to local optima, are able to obtain competitive solutions in reasonable computational times. For instance, see the work in [14] and [15].

iii) Finally, as an extension to heuristic approaches, metaheuristics provide a high-level problem-independent algorithmic framework that sets the guidelines to develop optimization algorithms. These approaches find near-optimal solutions by iteratively improving intermediate solutions with regard to a given measure of quality [16].

Based on the previous alternatives, different objectives have been addressed when optimizing the VFN-FGE. Minimizing the number of virtual network function instances mapped on top of the infrastructure [17], or maximizing the number of successfully embedded service requests [14] are a couple of these objectives. Other works deal with power-based placement proposals. For example, minimizing the power consumption via a genetic algorithm approach [18] or providing robustness to unknown or imprecisely formulated resource demand variations [2], are some of these works.

In addition to the previous references, Reinforcement Learning has become a promising research line to work on. For instance, Mijumbi et al. (2014) [19] used Q-learning to control the resources allocated as part of the NFV management system. In [20], Mijumbi also employed an artificial neural network to make resource reallocation decisions based on his previous work. Yao et al. (2018) [21] ulitized, for the first time, historical network request data and policy-based RL to optimize node mapping. They use an embedding representation of the node and links which, at each time step, are updated with changing features of the network attributes. This network embedding is passed through a convolutional neural network to select the substrate node that maximizes the long-term revenue.

B. Neural Combinatorial Optimization

In combinatorial optimization, a solution that either minimizes or maximizes a given cost function has to be identified from a finite set of solutions. When the cardinality of the search space of solutions is small, this task can be easy. Nevertheless, it is not straightforward on large problem instances, as exact methods are computationally inviable, and heuristics and metaheuristics are not guaranteed to find an optimal solution, nor to converge quickly. In Vinyals et al. (2015) [22], the authors proved that it is possible to solve combinatorial optimization problems using supervised learning. Particularly, they showed that, given the inputs for an instance, a neural network can return near optimal solutions for the problem. Motivated by recent progress in sequence-to-sequence models,

TABLE I
PROBLEM FORMALIZATION VARIABLES

H	set of hosts
L	set of links
V	set of VNFs
S	set of NS chains
R	set of resources
P	set of placements
a_{rh}	amount of resources r available in host h
r_{rv}	amount of resources r requested by VNF v
W_h^{min}	idle power consumption of host h
W_h^{cpu}	power consumption of each cpu in host h
W_{net}	power consumption per bandwidth unit on links
b_i	bandwidth of the link i
l_v	latency due to computation time of VNF v
l_i^s	latency on the link i produced by service chain s
b_v^s	bandwidth demanded by v in service chain s
l^s	maximum latency allowed on the service chain s
x_{fh}	binary placement variable for function f in host h
y_h	binary activation variable for host h
g_i	binary activation variable for link i

TABLE II
OPTIMIZATION PROBLEM FORMALIZATION EQUATIONS

$$\arg\min_{\boldsymbol{x}\in\Omega}\left(\sum_{h\in H}\left[W_h^{cpu}\cdot\sum_{f\in s}r_{rv}\cdot x_{fh}+W_h^{min}\cdot y_h\right]+\right.$$
$$\left.+\sum_{i\in L}W_{net}\cdot\sum_{f}b_v^s\cdot x_{fh}\right) \quad (1)$$

subject to:

$$\sum_{f\in s}r_{rv}\cdot x_{fh}\leq y_h\cdot a_{rh}\qquad\forall\, h\in H, r\in R \quad (2)$$

$$\sum_{f\in s}b_v^s\cdot x_{fh}\leq g_i\cdot b_i\qquad\forall\, i\in L \quad (3)$$

$$\sum_{h\in H}\sum_{f\in s}l_v\cdot x_{fh}+\sum_{i\in L}\sum_{f\in s}l_i^s\cdot x_{fh}\leq l^s\qquad\forall\, h\in H, i\in L \quad (4)$$

they developed an attentional Pointer Network to approach traditional combinatorial problems. However, in most of the cases, this strategy is not applicable since it is very hard to compute an optimal set of labels (solutions) to train the model.

Along the same line, Bello et al. (2016) [3] introduced the use of Reinforcement Learning to solve combinatorial problems, coining the term *Neural Combinatorial Optimization*. NCO relies on policy-based methods to directly learn a policy function that maps an instance of the problem (state) to a solution (action). When addressing combinatorial problems under the RL approach, the action space corresponds to the search space, which has a large dimensionality. In such a context, policy-based methods have proved to be effective when estimating the model parameters.

Using RL, optimal labels are not required to train the model, instead the rewards that the agent (name given to the model in RL) obtains from evaluating the solutions on the environment (in this case, the problem description) are used to optimize its policy. In particular, Policy Gradients [23] method is used to perform a stochastic gradient descent to better estimate the weights of the NN. Once the agent converges on the learning process, given an instance of the problem to the model, it returns a solution. This solution cannot be proven to be optimal, yet is "close" to it in the sense that it follows the policy that, on average, has obtained the best results.

The work here presented is related with Mirhoseini et al. (2017) [5], who also performs a placement model inspired by the original paper of Bello et al. [3]. In that paper, the authors implement an active search to infer a near optimal placement for a previously given instance of the problem. Our work differs in the sense that we handle constraints in the definition of the problem.

III. PROBLEM FORMALIZATION

In the following paragraphs, the mathematical formalization of the simplified VNF-FGE problem, together with the required notation, is described extensively.

Let us consider a list of network services that has to be optimally placed in a set of host servers $h \in H$, and each of those host servers relies on a limited amount of available resources $r \in R$, in terms of computing, storage and connection capabilities. As previously mentioned, the host servers are interconnected through a star topology using their own link connections $i \in L$ (see Fig. 1). Link attributes as bandwidth or propagation delay are also considered. The final purpose of this problem is to discover the optimal placement for a given service chain that minimizes the total power consumption of the infrastructure. This solution is subject to complying with the restrictions associated to the availability of virtual resources and link capacities, as well as the latency thresholds that each service imposes.

In order to formulate the problem, we take the nomenclature proposed by Marotta et al. (2017) [2]. Let us denote as $\{h_1, h_2, \ldots, h_n\}$ the set of host servers H, and let V be the set of VNFs available in the repository. So, a network service consists of an array of $m \in \{1, \ldots, M\}$ virtual network functions, that compose a service chain $s = (f_1, f_2, \ldots, f_m)$ where $f \in V$. The combinatorial space of all service chains is denoted as S.

The problem consists of finding the optimal set of placements denoted as $\boldsymbol{x} \in \{0,1\}^{m\times n}$, where x_{fh} stands for a boolean status variable that describes whether the function $f \in V$ is placed in host $h \in H$ or not (1 in the positive

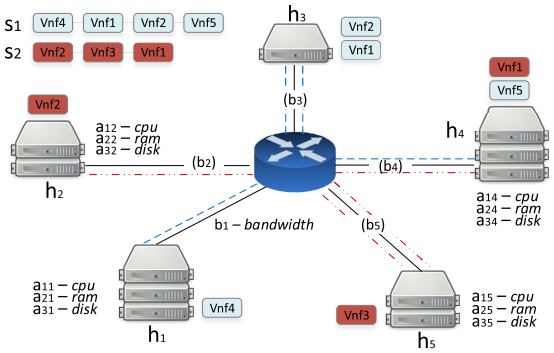

Fig. 1. Example of network service allocation over a virtualized environment. It depicts an environment in which the service chains, denoted as s_1 and s_2, need to be optimally placed in a total of five hosts (from h_1 to h_5). Each host has its own capacities for a_{1h} computation, a_{2h} memory and a_{3h} disk capacities. In addition, each host is connected to a common switch through a dedicated link i, which has an associated capacity bound b_i.

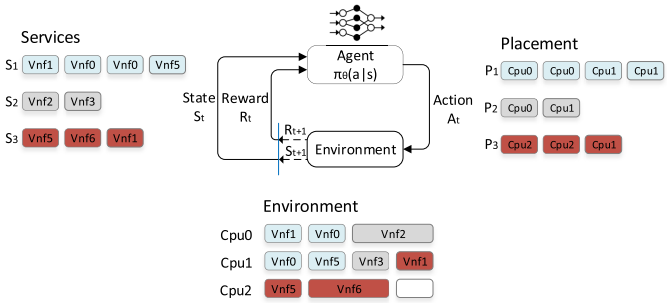

Fig. 2. Reinforcement learning action-reward feedback loop.

case, and 0 in the negative). Then, the search space in which the solution for the problem needs to be found is $\Omega = \{\boldsymbol{x} \in \{0,1\}^{m \times n} \text{ s.t. } \sum_h x_{fh} = 1 \; \forall f \in s\}$. The restriction in Ω states that a function can only be placed in one host at a time.

Prior to presenting the problem cost and restriction functions, a summary of the decision variables and parameters of the problem is presented in Table I. As stated previously, the set of variables to optimize are those that define the placement $\boldsymbol{x}$. To support the problem description, auxiliary variables are presented. This is the case of the server activation variables $y_h \in \{0,1\}$, which indicate 1 if the server is executing any VNF, and powered off otherwise; and link activation variables $g_i \in \{0,1\}$, which are equal to 1 if the link i is carrying traffic and 0 otherwise.

In relation to the power consumption, the host servers are characterized by a linear power profile that grows in proportion to the computing utilization. Each server activated ($y_i = 1$) consumes a minimum power W_h^{min}, and its power increases with the sum of the CPU demanded by the VNFs assigned to the server. Each cpu in use consumes W_h^{cpu} watts. Regarding links, they also have an energy cost associated. Is is calculated by multiplying the cost per bandwidth utilization, denoted as W_{net}, by the bandwidth utilized in each link. The available resources, $r \in R$, that each server h owns are denoted as a_{rh}. The amount of resources r needed by VNF v is indicated in r_{rv}. The bandwidth requested for the data transfer of VNF v, being part of service $s \in S$, is expressed as b_v^s. In the same way, l_i^s represents the latency in the link i due to the service s. And l_v the latency due to computation time of VNF v. The maximum bandwidth allowed in link i is represented as b_i. Finally, we denote the maximum latency allowed for each service chain s by l^s.

The cost function to optimize is presented in Table II. It represents the power consumption, and is calculated as the sum of the power consumption related to the activated servers and the aggregated cost of the active links. The constraint in Eq (2) determines that the total resources used in a server must not exceed the available ones a_{rh} in active servers. Therefore, it sets a link between server activation variables y_h and allocation variables x_{fh}; only the servers that host some VNF are active. Then, the capacity constraints for bandwidth are defined in Eq. (3). It uses link status variables g_i in the same way as host activation ones. They link the boolean status of links to the status of the node activation variables: if a link is used, then its end-nodes must be activated; if a node is not activated, then neither is the associated link. Finally, the constraints in Eq. (4) express the latency requirement for a network service $s \in S$, establishing that the aggregation of the latency over the links used in the graph and the latency due to computation time must meet the latency limit l^s for the service.

For the sake of illustrating the problem, an example is depicted in Fig. 1. Here, an equivalent representation for the placement is introduced: $p^s = (p_1, p_2, \ldots, p_m)$ where $p_i \in H$. This notation will be especially useful for simplifying the formulation of the RL equations that will be described hereafter. To continue, each server is connected to a common switch through a dedicated link i. The objective is to place the service chains (denoted as s_1 and s_2) minimizing the overall cost function. In this example, the placement vectors computed for each service are $p^{s_1} = (h_1, h_3, h_3, h_4)$ and $p^{s_2} = (h_2, h_5, h_4)$.

IV. Policy Optimization Approach

In this work, we approach a simplified VNF-FGE problem using the Neural Combinatorial Optimization paradigm. For this purpose, we use a neural network model to infer a placement policy. The neural architecture proposed is a sequence-to-sequence model based on an encoder-decoder structure, an architecture that has achieved outstanding results in sequence prediction. This works as follows, the agent[1] receives as input a network service $s = (f_1, f_2, .., f_m)$ of a variable size m and outputs a placement vector $p^s = (p_1, p_2, .., p_m)$ indicating the allocation for each VNF. The underlaying neural network, denoted by its weights θ, infers a policy $\pi_\theta(p^s|s)$ that generalizes a placement strategy for all possible service chain compositions that can be generated via a VNF dictionary.

The neural architecture proposed has the output space limited to a sequence, which we use to predict the nodes to

[1]From now on, we will use the term "agent", used in reinforcement learning, or "model" interchangeably to refer to the neural network that infers the placement policy.

be occupied by a service. Reducing the networking to a star connection is a simplification that enables us to utilize this well known architecture that has achieved outstanding results in tasks such as neural machine translation.

In Fig. 2, we depict the general reinforcement learning action-reward feedback loop particularized for our problem. In this case, the interactions with the environment are limited to a single step. This means that, for a given network service request (i.e., s_3 depicted in red in the picture), a state vector to input to the agent is created, embedding the state of the environment (previous allocated services s_1 and s_2) and the requested service. The agent generates the corresponding placement vector (action) indicating where the new service should be placed. The environment then evaluates the placement decision, and using Eq. (1) computes a feedback signal indicating the quality of the solution (reward). Notice that for the agent, the environment described by equations presented in Table II is a black box. And by interacting with it, the agent will discover a policy that solves the underneath constraint optimization problem without even knowing its definition.

As described, Neural Combinatorial Optimization cannot be directly applied, as it does not consider constraints within its machinery. Dealing with constraint dissatisfaction is essential, otherwise the cost function does not present enough information to infer a competitive policy. To that end, a novel reinforcement learning interface between the agent and the environment has been introduced. This interface enables us to use constraint relaxation techniques in the cost function of policy gradient method. Without this contribution, it is near impossible for the agent to improve its behaviour in constrained environments, as it would not experience enough positive rewards. The problem of sparsity reward is well known in Reinforcement Learning. In this paper, we extend neural combinatorial optimization theory to undertake this issue in constrained optimization problems. For that purpose, the reward signal indicating the energy consumption of the infrastructure is complemented with additional feedback signals, indicating the degree of constraint dissatisfaction. As we will see in the section below, these constraints are incorporated into the cost function using the Lagrange relaxation technique.

A. Constrained Optimization With Policy Gradients

As previously mentioned, we resort to Policy Gradients to learn the parameters of the stochastic policy $\pi_\theta(p^s|s)$ that, given as input a network service $s \in S$, assigns high probabilities to placements $p^s \in P$ with lower costs, and low probabilities to those with higher costs[2]. Our neural network uses the chain rule to factorize this output probability (see Eq. 1). The likelihood of allocating a VNF in the sequence therefore depends on the previously allocated VNFs of the requested chain $p_{(<f)}$ and the state vector. For simplicity, we will consider that the state vector is fully defined by the network service s. If multiple states of the environment were taken into consideration, the state to input the agent would be an embedding of the service in conjunction with the state of the environment.

$$\pi_\theta(p|s) = \prod_{f=1}^{m} \pi_\theta(p_f \mid p_{(<f)}, s) \tag{1}$$

To estimate the parameters of the model, Policy Gradients define an objective function that represents the expected reward achieved for every vector of weights θ. It determines the quality of the policy achieved, and is therefore the function to optimize. This objective function is defined for every possible policy, and its shape depends on both the environment and the NN model designed. Subsequently, the problem does not rely on directly optimizing the Eq. (1), but on optimizing this new objective function that determines the policy of the agent. For this purpose, let us start defining the expected energy consumption E associated to a placement given an input network service s:

$$J_E^\pi(\theta|s) = \mathop{\mathbb{E}}_{p\sim\pi_\theta(\cdot|s)}[E(p)] \tag{2}$$

The agent needs to infer a policy to place services from all possible service compositions. Hence, the expected energy cost is defined as an expectation from the service distribution:

$$J_E^\pi(\theta) = \mathop{\mathbb{E}}_{s\sim\mathcal{S}}[J_E^\pi(\theta|s)] \tag{3}$$

Also, there is an expectation of constraint dissatisfaction associated to a policy that can be expressed as follows:

$$J_C^\pi(\theta) = \mathop{\mathbb{E}}_{s\sim\mathcal{S}}[J_C^\pi(\theta|s)] \tag{4}$$

The primal problem becomes to find the policy that minimizes the expected energy consumption subject to the satisfactions of the constraints:

$$\min_{\pi\sim\Pi} J_E^\pi(\theta) \quad \text{s.t.} \quad J_{C_i}^\pi \leq 0 \tag{5}$$

We define a function J_C^π for every constraint dissatisfaction signal that the environment returns. In our case, there are three of these signals, and they represent the cumulative constraint dissatisfaction grouped by occupancy, bandwidth and latency.

Using the Lagrange relaxation technique, the problem represented in Eq. (5) is converted into an unconstrained problem where unfeasible solutions are penalized [7]:

$$\begin{aligned} g(\lambda) = \min_\theta J_L^\pi(\lambda,\theta) &= \min_\theta [J_E^\pi(\theta) + \sum_i \lambda_i \cdot J_{C_i}^\pi(\theta)] \\ &= \min_\theta [J_E^\pi(\theta) + J_\xi^\pi(\theta)] \end{aligned} \tag{6}$$

being $J_L^\pi(\lambda,\theta)$ the Lagrangian objective function, $g(\lambda)$ the Lagrange dual function, and λ_i the Lagrange multipliers, in other words, penalty coefficients. In this equation we introduce a new term $J_\xi^\pi(\theta)$ we are going to refer to as the expected penalization, which is just the weighted sum of all expectation of constraint dissatisfaction signals.

The dual function is convex, even though the primal function and the constraints are non-convex, and it gives a lower bound on the optimal value for the primal problem (Bertesekas, 1999) [24]. The objective then is to find the Lagrange coefficients that produce the best lower bound. This is known as the Lagrange dual problem:

$$\max_\lambda g(\lambda) = \max_\lambda \min_\theta J_L^\pi(\lambda,\theta) \tag{7}$$

[2]For the simplicity of notation we will refer to the placement vector p^s as p.

In this case, we perform a manual selection of the penalty coefficients. An intuition on how these coefficients are obtained is described in Appendix C. The resulting Lagrangian function $J_L^\pi(\theta)$ is the objective function that defines the quality of the policy inferred.

In order to compute the weights θ that optimize this objective function, we resort to Monte-Carlo Policy Gradients and stochastic gradient descent:

$$\theta_{k+1} = \theta_k + \alpha \cdot \nabla_\theta J_L^\pi(\theta) \tag{8}$$

The gradient of the Lagrangian is derived using the log-likelihood method. This derivation process does not present any difference with respect to deriving the expected reward, method introduced in (Williams, 1992) [25].

$$\begin{aligned} \nabla_\theta J_L^\pi(\theta) &= \mathop{\mathbb{E}}_{p\sim\pi_\theta(\cdot|s)}[(L(p|s)\cdot\nabla_\theta \log \pi_\theta(p|s)] \\ \text{where} \quad L(p|s) &= E(p|s) + \xi(p|s) \\ &= E(p|s) + \sum_i \lambda_i \cdot C_i(p|s) \end{aligned} \tag{9}$$

We denote the penalized energy cost obtained in each iteration as $L(p|s)$, which is calculated by adding to the energy signal $E(p|s)$ the weighed sum of all the constrained dissatisfaction signals $C(p|s)$.

The gradient is then approximated with Monte-Carlo sampling, where B sample services are drawn $s_1, s_2, \ldots, s_B \sim \mathcal{S}$. To reduce the variance of the gradients, and therefore to faster convergence, we include a baseline estimator [23], indicated with the term $b(s)$ in the formula.

$$\nabla_\theta J_L^\pi(\theta) \approx \frac{1}{B}\sum_{j=1}^{B}(L(p_j|s_j) - b_{\theta_\nu}(s_j))\cdot\nabla_\theta \log \pi_\theta(p_j|s_j) \tag{10}$$

The baseline used in this case is state dependant and is performed by an auxiliary sequence network that embeds the state information and predicts the penalized cost $L(p|s)$ that the agent obtains following the current policy $\pi_\theta(p|s)$. It is parametrized by the weights θ_ν and is trained with stochastic gradient descent on the mean squared error objective between its predictions $b_{\theta_\nu}(s)$ and the actual penalized cost obtained from the environment.

$$\mathcal{L}(\theta_\nu) = \frac{1}{B}\sum_{j=1}^{B}\|b_{\theta_\nu}(s_j) - L(p_j|s_j)\|^2 \tag{11}$$

The algorithm implementation of the single time-step Monte-Carlo Policy Gradients with a baseline estimator is described in Appendix D.

B. Architecture Details

In the NCO approach we propose, the agent is a sequence-to-sequence model (Sutskever et al. 2014) [26]. Fig. 3 shows the overall agent architecture, which is formed by an encoder-decoder design based on stacked Long Short-term Memory (LSTM) cells.

The input to this model is the sequence of VNFs that composes the network service to be placed $s = (f_1, f_2, \ldots, f_m)$.

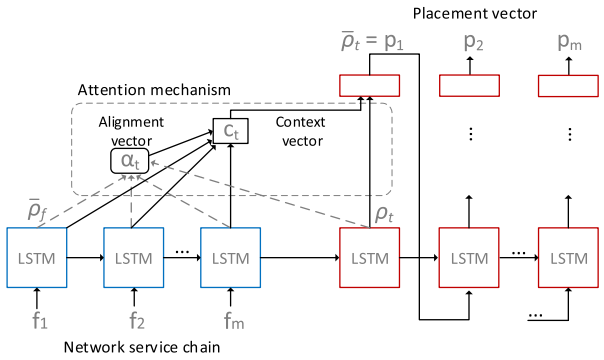

Fig. 3. Sequence-to-sequence model composed of an encoder-decoder architecture with a Bahdanau attentional mechanism. It encodes a variable length service chain in a representational vector that the decoder network uses to produce the placement decision.

As described, these NS chains have a variable length $m \in \{1, \ldots, M\}$. This is the main reason for using a sequence model, as it is designed to input chains of different sizes without modifying its internals.

The decoder is an attentional LSTM model (Bahdanau et al. 2015) [27] which has the same number of decoding steps as the input sequence. At each step, the decoder outputs the host to place the component introduced in the encoder at the same step. The decoder network hidden state $\rho_t = f(\rho_{t-1}, \bar{\rho}_{t-1}, c_t)$ is a function of its own previous state combined with an attention over the encoder hidden states. The context vector c_t consists of the sum of hidden states of the input sequence, weighted by alignment scores. The context vector for the decoder output step t is calculated as follows:

$$c_t = \sum_f \alpha_{t,f}\bar{\rho}_f \tag{12}$$

The set $\alpha_{t,f}$ are the variables defining the weight of each source hidden state in the decoding process. The variable-size alignment vector has the same number of steps as the source sequence. It is computed by scoring the current target hidden state of the decoder ρ_t with each source hidden state $\bar{\rho}_f$:

$$\alpha_{t,f} = \text{softmax}(\text{score}(\rho_t, \bar{\rho}_f)) \tag{13}$$

The score function is defined in the following form:

$$\text{score}(\rho_t, \bar{\rho}_f) = v_a^\intercal \tanh(W_1\rho_t + W_2\bar{\rho}_f) \tag{14}$$

being v_a and W_1, W_2 the weight matrices to be learned in the alignment model.

Finally, the baseline $b_{\theta_\nu}(s)$ is computed by an auxiliary network; an LSTM encoder connected to a multilayer perceptron (MLP) output layer that predicts the penalized energy cost that the agent produces following the current policy. It is, therefore, a value approximator based on the environmental state.

C. Search Strategies

The Policy Gradients method presents a major drawback, it suffers from local convergence. During the learning process, the weights of the neurons are adjusted in the direction of the gradient of the objective function. As this function is non-convex, this approach is prone to converging to sub-optimal

TABLE III
SUMMARY OF THE PROBLEM PARAMETERS, MODEL SETTINGS AND EXPERIMENTAL RESULTS

Service length	Model	No. Hosts	Encoder/Decoder Layers x LSTM size	No. parameters	Occupancy error ratio	Bandwidth error ratio	Latency error ratio	Placement error ratio
12	Small	10	1x32	22015	5.4%	0.0%	0.3%	5.7%
14	Small	10	1x32	22015	22.0%	19.3%	0.0%	35.3%
16	Small	10	1x32	22015	34.5%	72.5%	0.0%	79.5%
18	Small	10	1x32	22015	63.9%	100.0%	0.0%	100.0%
20	Large	20	3x64	213577	2.1%	0.0%	0.0%	2.1%
24	Large	20	3x64	213577	13.2%	0.0%	0.0%	13.2%
28	Large	20	3x64	213577	71.5%	0.7%	0.0%	71.5%
30	Large	20	4x128	1099849	94.0%	2.9%	1.8%	94.9%

minima. Once the agent converges to one, it preserves the achieved policy no matter how long we extend the training. To improve the policy obtained, the convergence to a better optima in the cost function is necessary. In this sense, techniques such as entropy regularization are applied during training to increase exploration.

We have obtained significant improvements by applying some of the search methods at inference as presented in Bello et al. [3]. As evaluating the placement is inexpensive, the agent can simulate a search procedure at inference time by considering multiple candidate solutions and selecting the best one. We have considered two search strategies: a greedy inference over multiple trained models, and a sampling technique. In the first one, multiple models are learned, and at inference time the greedy output from every model is evaluated to select the best one. Another alternative consists of improving the exploration at inference using a temperature hyperparameter T to control the sparsity of the output distribution. Multiple samples are taken and the best of them is selected as the output. This method is used in order to prevent the model from being overconfident, allowing to evaluate at inference proximal policies.

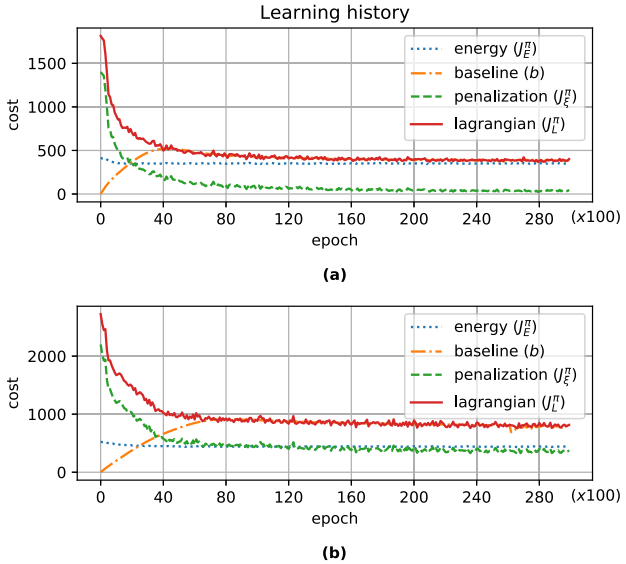

Fig. 4. Learning history on a small problem instance with **(a)** sufficient space available in the infrastructure, and **(b)** a high occupancy ratio.

V. EXPERIMENTAL STUDY

In this section, in order to evaluate the proposed approach for optimizing the VNF-FGE problem, a detailed experimental study is carried out in two environments of different sizes: a small and a large infrastructure. Using this testbed, we evaluate the performance of the neural network model against *Gecode* solver and the FF heuristic algorithm [28]. Those environments have been selected to benefit the solver and heuristic respectively as argued below.

In addition to directly using the output of the neural network, we have also employed this information to guide the heuristic algorithm. The resulting hybrid agent utilizes the outcome from the neural network to indicate to the heuristic the order in which the nodes in the infrastructure must be occupied.

A brief description of the environments, together with a summary of the main parameters[3] and results is available in Table III. For further information on the environment or parameter configuration refer to Appendices A and B.

[3]The model together with the weights to reproduce the results obtained in the experimental study is available in: https://github.com/rubensolozabal/vnf_placement_optimization_rl

A. Learning Process

First, we conducted experiments in order to study the learning process of the bare sequence-to-sequence model for an instance where different occupancy ratios are considered.

In Fig. 4, the learning history of the agent on a small instance of the problem (a) with sufficient space available and (b) with a substantial occupancy ratio is presented. For each iteration, the approximation calculated in that batch for the expected energy J_E^π, baseline b, penalisation J_ξ^π, and Lagrangian J_L^π functions are introduced.

As shown, at the beginning of the learning, the agent generates random output sequences that violate many constraints, receiving a high penalty for these actions. Therefore, at the beginning, the agent only focuses on constraint satisfaction and ignores the underlying power consumption. As learning progresses, the agent corrects its weights via stochastic gradient descent to minimize the Lagrangian objective function (Eq. 6). Repeating this process iteratively, the agent improves its policy, in other words, reduces the constraints dissatisfied. This process continues until a local minima or a saddle point is reached.

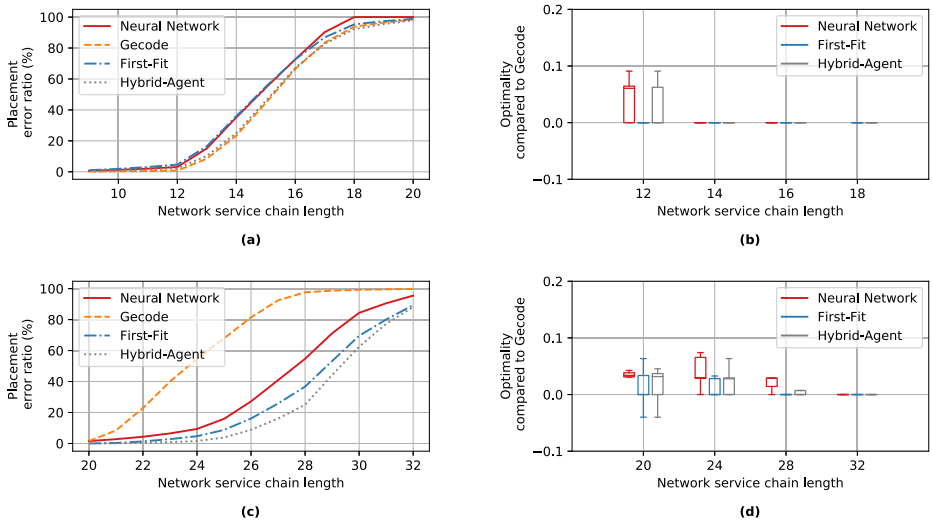

Fig. 5. The solution validity comparison in terms of optimality and feasibility between the neural network model, Gecode solver, the First-Fit heuristic and the Hybrid-Agent in a small **(a)** and **(b)** and a large **(c)** and **(d)** environment.

As depicted in Fig. 4(a), at the end of the learning, the penalty signal almost cancels. This means that the policy that the agent has been able to infer in this small scenario is that which is desired. The expectation of obtaining a placement that does not meet the restrictions is very low, and if this situation occurs, the constraints are going to be slightly dissatisfied. However, in a tightly constrained environment, Fig. 4(b), the probability of constraint dissatisfaction increases significantly. As can be seen, the mean penalty received is comparable to the energy cost. Therefore, the agent focuses on satisfying the constraints in the way that the penalty obtained is minimized.

B. Evaluating Search Strategies

As pointed out in Section IV-C, in order to improve the results obtained in the learning process, we have conducted the following search methods at inference: greedy inference over multiple agents and a sampling method. The method that has produced the best results is the greedy inference over multiple trained agents. Here, each agent trained separately converges to its own policy, leading to the same input sequence for different placement strategies. Selecting the best solution from multiple agents decisions leads to a significant improvement. This technique has been applied in combination with a sampling method, which in our case has not resulted as efficient as in [3]. The results obtained after applying these techniques are reflected in Table III.

C. Comparison With Other Algorithms

For the purpose of validating the proposed NCO approach, we compare the results that the model produces in real-time with the output of the constraint solver *Gecode*[4] and the FF heuristic. This experiment is performed restricting the time allowed to the solver up to 5 minutes. For each instance of the VNF-FGE (small and large), we evaluate the results in terms of the number of feasible solutions and also optimality when compared to the solver. The detail of each problem instance as the summary of the model parameters used in each case can be seen in Appendix B.

Fig. 5 shows the indicators mentioned in a small **(a)** and **(b)**, and a large **(c)** and **(d)** environment obtained by the neural network model, the solver and the FF heuristic. This experiment has been measured under the following procedure, starting from an empty environment, we locate a single service which has a different length in order to tighten the occupancy, therefore hardening the constraints that define the problem. This testbed neither represents the normal operation, where services are sequentially allocated, nor is their length realistic. Nevertheless, it has been considered for comparison purposes. The same effect can be achieved considering a more restricted environment, yet this method helps unify the starting point for the tests. As can be noted, we increase the number of VNFs in the chain, in other words, we constrain the space of valid solutions, until all methods fail. For small instances of the problem, the solver has a greater probability of finding a feasible solution. However, in large problem instances, the ratio of failures turns around, the heuristic being that suffers fewer misstatements.

With regard to the optimality gap, for those instances in which the solution satisfies all constraints, we have compared the quality of the solution between the solver, and the neural network and the heuristic algorithm respectively. The solver obtains better results when the environment is small, but again it is beaten by the heuristic when tested in a large enough environment. The neural network on its own performs in between the methods mentioned. It provides better solutions than the solver in the second scenario (achieving up to 50% more feasible solutions when the service length is between 24 and 26 elements), but it is less competitive than the heuristic in both the number of feasible solutions and the optimality gap.

[4]According to the guidelines in [8], the Gecode solver implements an algorithm that is based on the classical Branch & Bound paradigm.

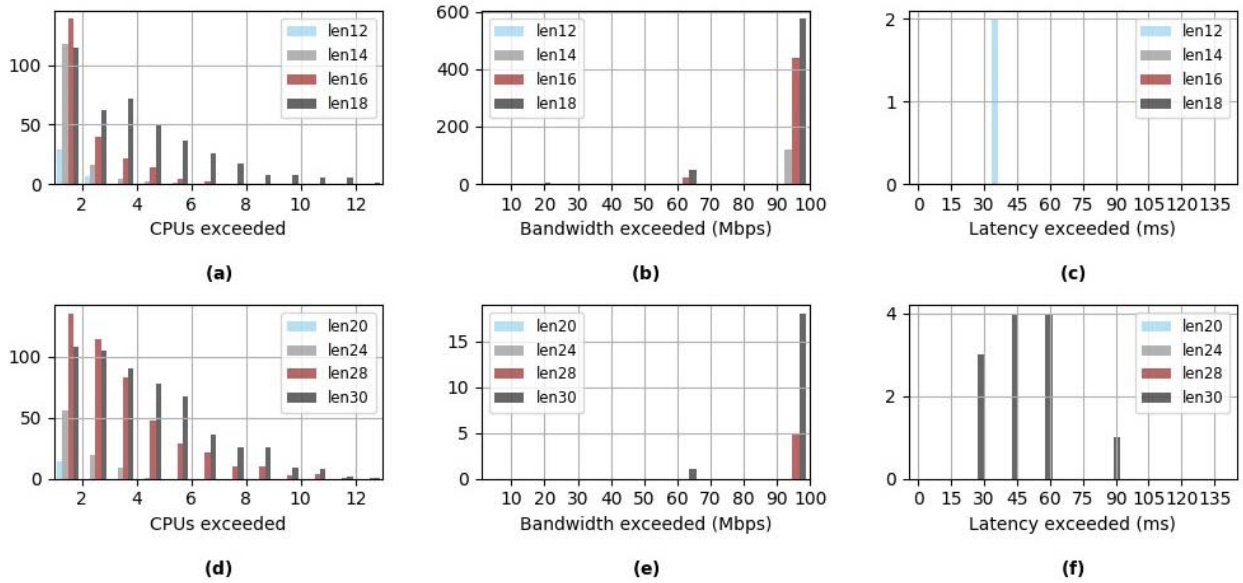

Fig. 6. Histogram representation of the number of CPUs, bandwidth (Mbps) and latency (ms) exceeded, in that order, along placements of different length services in a small **(a) (b) (c)** and a large **(d) (e) (f)** environment.

Using a neural network to estimate the policy presents a great asset, it always produces in real-time a solution that approximates the result while at the same time trying to minimize the constraints dissatisfied. However, in the case of failure using, for example, the solver, no solution is available. But this behaviour can also be obtained in a heuristic.

D. Analysis Hybrid-Agent

As the results show, in real scenarios the FF heuristic on its own is able to produce better solutions when compared with the neural network and the solver. Nevertheless, this algorithm lacks customization and for each problem instance it performs a deterministic behaviour. It assigns the environment starting with the nodes that have more resources available. The information that the neural network produces can be used as a guidance to improve the performance of this heuristic algorithm. The resulting hybrid agent has shown to enhance the number of valid solutions in highly constrained environments while maintaining the optimality gap of the heuristic.

The Hybrid-Agent has been designed as follows. It uses the output sequence produced by the neural network as a seed for the FF algorithm. The output produced by the neural network guides the heuristic indicating the server that must be occupied. Nevertheless, the heuristic verifies that the immediate location is allowed, otherwise it jumps to occupy the next server the neural network suggests. Using this procedure, we achieve an agent that becomes the new reference. As depicted in Fig. 5, its solutions are comparable to the solver in small environments, and it outperforms the all previous methods in the number of valid solutions when the environment is large enough. In that case, we obtain a consistent 10% benefit in the number of feasible solutions compared to the heuristic, while preserving its optimality gap.

E. Constraint Dissatisfaction Pattern

As previously discussed, the neural network infers a policy that locally minimizes the objective function. This is, the policy achieved is the one that, for all combinatorial space of services S, presents a local minimum in the expectation of energy consumption, including the penalization that misplacements cause. As can be seen, when there is sufficient space available in the infrastructure, the neural network is able to generalise placement decisions satisfying all constraints. As the environment occupancy increases, the policy is not sophisticated enough and starts to dissatisfy the constraints that suppose the least penalisation. The way in that these constraints are dissatisfied follows a pattern, in the sense that dissatisfying a variable by little is more probable than doing so by a lot. A priori, is not possible to know what constraints are going to be dissatisfied.

In Fig. 6, the histogram of the dissatisfied constraints grouped by occupancy, bandwidth and latency is depicted. The histograms are generated from a set of 640 samples each, on the previously described scenarios. As can be seen, depending on the instance, the agent is prone to violating some constraints before others. For example, despite occupancy being generally the most dissatisfied constraint, the specific case of bandwidth constraints in the small scenario **(b)**, when the services have a considerable length (18 elements), is the main cause of dissatisfaction. Neither the constraints dissatisfied nor the dissatisfaction pattern can be predicted. Yet it is possible to change that pattern by modifying the assigned Lagrange multiplier λ_i. The agent can be biased in order to meet desired constraints prior to others.

This behaviour, cannot be obtained nor transferred to a heuristic. As in this case, the dissatisfaction pattern is embedded in the algorithm itself.

VI. Conclusions

In this work, we solve the VNF-FGE problem by means of a Reinforcement Learning approach to model a placement policy in a Network Function Virtualization infrastructure. For that purpose, we extend *Neural Combinatorial Optimization* theory in order to consider restrictions in the problem. The resulting neural network model is able to learn placement decisions, with the aim of minimizing the overall power consumption.

Conducted experiments have demonstrated that when the proposed agent is used in conjunction with a heuristic algorithm, we obtain a hybrid agent that improves the performance of the heuristic itself. In this paper, artificial intelligence has been used to enhance a simple First-Fit heuristic, achieving competitive results without requiring expertise in the problem.

An exciting extension to improve the policy the neural network acquires could be presenting the problem as a fully defined Markov Decision Process. In that case, as placement decisions are made step-by-step, the agent is able to learn from intermediary states, instead of inferencing it at once from a single input. This is the strategy that Nazari et al. (2018) [6] has followed to solve the Vehicle Routing Problem. In their model, they do not require excessive post processing at inference to obtain solutions that are comparable in running time and performance to problem specific heuristics.

Appendix A
Environment Description

Description of the environment referenced as *small* during the paper:

Host Properties										
No. CPUs	10	9	8	7	6	6	6	6	6	6
Link BW (Mbps)	1000	1000	500	400	300	300	300	300	300	300
Link Latency (ms)	30	50	10	50	50	50	50	50	50	50

Description of the environment referenced as *large* during the paper:

Host Properties										
No. CPUs	10	10	9	9	8	8	7	7	6	6
Link Bw (Mbps)	1000	1000	1000	1000	500	500	400	400	300	300
Link Latency (ms)	30	30	50	50	10	10	50	50	50	50
	6	6	6	6	6	6	6	6	6	6
	300	300	300	300	300	300	300	300	300	300
	50	50	50	50	50	50	50	50	50	50

Description of the VNF Dictionary:

VNFD Properties								
No. CPUs required	4	3	3	2	2	2	1	1
Bw required (Mbps)	100	80	60	20	20	20	20	20
Processing latency (ms)	100	80	60	20	20	20	20	20

The parameters related to power consumption in the environment are the following:

W_h^{min}	200 *(watt)*
W_h^{cpu}	100 *(watt)*
W_{net}	0.1 *(watt / mbps)*

The Lagrange multipliers selected to penalize the different constraint functions are presented below:

$\lambda_{occupacy}$	1000
$\lambda_{bandwidth}$	10
$\lambda_{latency}$	10

Appendix B
Hyperparameters

The main hyperparameters configured in the model are presented in the following table:

Hyperparameter	Value
Learning rate (agent)	0.0001
Batch size	128
No. LSTM layers	(1, 3, 4)*
LSTM hidden size	(32, 64, 128)*
Embedding size	10
Learning rate (baseline)	0.1
Gradient clipped by norm	1
Temperature	15
No. samples with temp.	16
No. models inference	6
*Refer to Table III to consult in which scenario the different dimensionalities where used.	

During the experimentation and tuning of the architecture, we have observed that the complexity of the encoder-decoder structure needs to be sufficiently large to embed the features of the problem. Yet, once this requirement is satisfied, increasing its complexity does not result in better performance. In that case, decreasing the learning rate, increasing the batch size and encouraging exploration leads to better results. In this paper, we have focused on enhancing the solutions evaluating some search strategies at inference. We have evaluated up to a total of 6 different models on which 16 samples where taken over the smothered output distribution using a temperature parameter. These post-processing techniques used at inference do not add significant computation time and provide significant improvement. In our experience, sampling from a slightly smothered distribution is the technique that has reported the best results. Inferring from multiple models, on its own, has also benefited, yet a significant correlation between the policies inferred exists. Therefore, there is a commitment between the number of models used and the improvement experienced.

Appendix C
Intuition on the Selection of the Lagrange Coefficients

In this appendix we aim to provide some intuitions on the selection of the Lagrange multipliers. Let us refer to the optimal of the primal problem described in Eq. (5) as p^*, the minimum value that the objective function can obtain subject to the satisfaction of the constraints. Using the Lagrange relaxation technique, this problem is transformed into an unconstrained problem where unfeasible solutions are penalized, see Eq. (6). The resulting dual function $g(\lambda)$ is always convex, even though the primal function and the constraints are non-convex, and it gives a lower bound on the optimal value for the primal problem p^*.

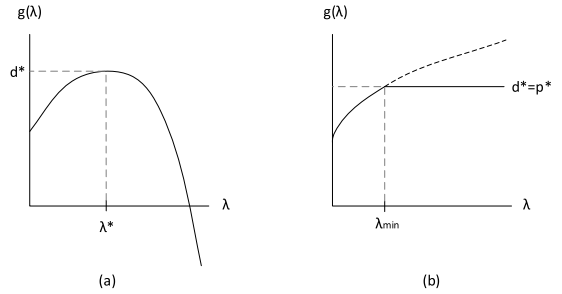

Fig. 7. General dual function representation in cases where: **(a)** a benefit in the feasible regions exists, **(b)** no subsidy is received in the feasible region or a feasible region does not exist.

In general, the objective is to find the Lagrange coefficients that produce the best lower bound d^*. This is known as the Lagrange dual problem and, for a general case, it is represented in Fig. 7a.

The problem we describe presents some peculiarities. All the constraint functions are positive definite, because they are defined as the expectation of occupancy, bandwidth and latency dissatisfaction respectively. In other words, a feasible solution would be the one in which all those expectation of constraint dissatisfaction signals were equal to zero. Nevertheless, there might not exist in the whole space of weights θ a configuration of the neural network that fulfils this requirement. If it exists, then $d^* = p^*$. Strong duality holds because there exists a point that deletes the penalty term (complementary slackness).

In general, a feasible point cannot be proven to exist. In that case, the dual function constantly increases with λ, as every point of the function is more and more penalized. Consequently, $d^* = \infty$.

These two scenarios are depicted in Fig. 7b. In the first one, there exists a λ_{min} at which point we can guarantee that the penalization is large enough to affirm that the optima of the dual function d^* corresponds to the optima of the primal problem p^*. This value is a priory unknown but can be easily estimated. It might seem that all values for λ between λ_{min} and ∞ are equally valid, but we will clarify this later.

In the second case, the objective is different. A primal optima p^* does not exist. The goal is to find the Lagrange coefficients λ that establish the desired commitment between the expectation of a penalty J_C^π and the expectation of reward J_E^π. To clarify this concept, let us analyze the extreme values for λ. If $\lambda = 0$, then the Lagrangian $J_L^\pi = J_E^\pi$, the optimizer is going to minimize the expected power consumption without paying attention to the constraints. Similarly, if $\lambda \to \infty$, by the law of big numbers, the Lagrangian is going to converge to the penalization function J_C^π. Therefore, the policy that the agent is going to infer is the one that achieves by mean fewer misstatements without taking into account the optimality of the solution. In this case, there is no λ that maximizes the dual function $g(\lambda)$ to give us a reference point on what penalization is enough in order to the feasible region, or in this case, the region that presents a minimum in constraint dissatisfaction, surpassing the rest of the function.

In practice, if the neural network is a good enough approximator, then a weight configuration exists in which $J_C^\pi \approx 0$. This gives us an intuition that the region to find the Lagrange coefficients is λ_{min}^+. Once the penalty coefficients are set, we obtain the Lagrangian function $J_L^\pi(\theta)$ to optimize. This Lagrangian is a non-convex function and, therefore, it cannot be guaranteed that the optimizer is going to achieve the global optima. Normally this function is optimized until a saddle point or a local optima is reached. A priori, we do not have information on the convergence point and a fine tune of the λ is going to be needed to set the desired commitment between optimality and constraint dissatisfaction. Notice that during this fine tuning we move by following the trajectory that joins the local optima or saddle point achieved during the learning in the extreme cases ($\lambda = 0$ and $\lambda = \infty$). In theory, we should experience benefits in favor of the goal we want to achieve by controlling the λ. Yet, in reality, nothing guarantees that during this procedure a different optimization path with a better or worse performance is achieved.

APPENDIX D
ALGORITHM IMPLEMENTATION

Implementation of the single time-step Monte-Carlo Policy Gradients with baseline estimator:

Algorithm 1

procedure TRAIN AGENT NETWORK (LEARNING SET $\mathcal{S}$, BATCH SIZE B)
  Initialize agent and critic networks with random weights θ and θ_ν
  for epoch = 1,2... **do**
    reset gradients: $d\theta \leftarrow 0$
    $s_j \sim \text{SampleInput}(\mathcal{S})$ for $j \in \{1, .., B\}$
    $p_j \sim \text{SampleSolution}(\pi_\theta(\cdot|s))$ for $j \in \{1, .., B\}$
    $b_j \leftarrow b_{\theta_v}(s_j)$ for $j \in \{1, .., B\}$
    compute cost function : $L(p_j)$ for $j \in \{1, .., B\}$
    $g_\theta = 1/B \cdot \sum_{j=1}^{B} (L(p_j) - b(s_j)) \cdot \nabla_\theta \log \pi_\theta(p_j|s_j)$
    $\mathcal{L}(\theta_\nu) = 1/B \cdot \sum_{j=1}^{B} \|b_{\theta_\nu}(s_j) - L(p_j|s_j)\|^2$
    $\theta \leftarrow \text{Adam}(\theta, g_\theta)$
    $\theta_\nu \leftarrow \text{Adam}(\theta_\nu, \mathcal{L}(\theta_\nu))$
  end for
  return θ and θ_ν
end procedure

REFERENCES

[1] *Network Functions Virtualisation (NFV); Architectural Framework*, document ETSI GS NFV 002 (V1.2.1), 2014.

[2] A. Marotta, F. D'Andreagiovanni, A. Kassler, and E. Zola, "On the energy cost of robustness for green virtual network function placement in 5G virtualized infrastructures," *Comput. Netw.*, vol. 125, pp. 64–75, Oct. 2017.

[3] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," 2016, *arXiv: 1611.09940*. [Online]. Available: https://arxiv.org/abs/1611.09940

[4] M. Deudon, P. Cournut, A. Lacoste, Y. Adulyasak, and L.-M. Rousseau, "Learning heuristics for the TSP by policy gradient," in *Proc. Int. Conf. Integr. Constraint Program., Artif. Intell., Oper. Res.* Amsterdam, The Netherlands: Springer, 2018, pp. 170–181.

[5] A. Mirhoseini *et al.*, "Device placement optimization with Reinforcement Learning," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 2430–2439.
[6] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takác, "Reinforcement learning for solving the vehicle routing problem," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 9839–9849.
[7] C. Tessler, D. J. Mankowitz, and S. Mannor, "Reward constrained policy optimization," 2018, *arXiv:1805.11074*. [Online]. Available: https://arxiv.org/abs/1805.11074
[8] C. Schulte, M. Lagerkvist, and G. Tack, *Gecode*, 2006, pp. 11–13. [Online]. Available: http://www.gecode.org
[9] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *Proc. IEEE 4th Int. Conf. Cloud Netw. (CloudNet)*, Oct. 2015, pp. 171–177.
[10] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions," in *Proc. 11th Int. Conf. Netw. Service Manage. (CNSM)*, Nov. 2015, pp. 50–56.
[11] H. Moens and F. De Turck, "VNF-P: A model for efficient placement of virtualized network functions," in *Proc. 10th Int. Conf. Netw. Service Manage. (CNSM)*, Nov. 2014, pp. 418–423.
[12] A. Gupta, M. F. Habib, P. Chowdhury, M. Tornatore, and B. Mukherjee, "On service chaining using virtual network functions in network-enabled Cloud systems," in *Proc. IEEE Int. Conf. Adv. Netw. Telecommun. Syst. (ANTS)*, Dec. 2015, pp. 1–3.
[13] M. Shifrin, E. Biton, and O. Gurewitz, "Optimal control of VNF deployment and scheduling," in *Proc. IEEE Int. Conf. Sci. Elect. Eng. (ICSEE)*, Nov. 2016, pp. 1–5.
[14] R. Riggio, A. Bradai, T. Rasheed, J. Schulz-Zander, S. Kuklinski, and T. Ahmed, "Virtual network functions orchestration in wireless networks," in *Proc. 11th Int. Conf. Netw. Service Manage. (CNSM)*, Nov. 2015, pp. 108–116.
[15] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr./May 2015, pp. 1346–1354.
[16] R. Mijumbi, "Placement and scheduling of functions in network function virtualization," 2015, *arXiv:1512.00217*. [Online]. Available: https://arxiv.org/abs/1512.00217
[17] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions," in *Proc. IEEE Int. Symp. Integr. Netw. Manage.*, May 2015, pp. 98–106.
[18] S. Kim, S. Park, Y. Kim, S. Kim, and K. Lee, "VNF-EQ: Dynamic placement of virtual network functions for energy efficiency and QoS guarantee in NFV," *Cluster Comput.*, vol. 20, no. 3, pp. 2107–2117, 2017.
[19] R. Mijumbi, J.-L. Gorricho, J. Serrat, M. Claeys, F. De Turck, and S. Latré, "Design and evaluation of learning algorithms for dynamic resource management in virtual networks," in *Proc. IEEE Netw. Oper. Manage. Symp. (NOMS)*, May 2014, pp. 1–9.
[20] R. Mijumbi, J.-L. Gorricho, J. Serrat, M. Claeys, J. Famaey, and F. De Turck, "Neural network-based autonomous allocation of resources in virtual networks," in *Proc. Eur. Conf. Netw. Commun. (EuCNC)*, Jun. 2014, pp. 1–6.
[21] H. Yao, B. Zhang, P. Zhang, S. Wu, C. Jiang, and S. Guo, "RDAM: A reinforcement learning based dynamic attribute matrix representation for virtual network embedding," *IEEE Trans. Emerg. Topics Comput.*, to be published.
[22] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2015, pp. 2692–2700.
[23] R. S. Sutton *et al.*, *Introduction to Reinforcement Learning*, vol. 135. Cambridge, MA, USA: MIT Press, 1998.
[24] D. P. Bertsekas, "Nonlinear Programming," *J. Oper. Res. Soc.*, vol. 48, no. 3, p. 334, 1997.
[25] R. J. Williams, "Simple statistical gradient-following algorithms for Connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, 1992.
[26] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2014, pp. 3104–3112.
[27] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by jointly learning to align and translate," 2014, *arXiv:1409.0473*. [Online]. Available: https://arxiv.org/abs/1409.0473
[28] S. Kumaraswamy and M. K. Nair, "Bin packing algorithms for virtual machine placement in cloud computing: A review," *Int. J. Elect. Comput. Eng.*, vol. 9, no. 1, p. 512, 2019.

Ruben Solozabal received the B.S. and M.Sc. degrees in telecommunications engineering from the University of the Basque Country in 2015. He is currently pursuing the Ph.D. degree with particular interest in reinforcement learning. After working in the automation industry for three years focusing on industrial communications, he decided to join Academia. His work is focused on designing real-time decision making systems to optimize 5G networks.

Josu Ceberio received the bachelor's and master's degrees in computer science from the University of the Basque Country in 2007 and the Ph.D. degree in computer science from the Intelligent Systems Group in 2014. Since 2010, he has been a member of the Intelligent Systems Group. Since 2014, he has been a Lecturer with the University of the Basque Country, where he is currently with the Department of Computer Science and Artificial Intelligence, Faculty of Computer Science. He has coauthored more than 35 scientific publications in different journals and international conferences covering topics, such as permutation-based combinatorial optimization problems, estimation of distribution algorithms, elementary landscape decomposition, and neural networks. He has also actively participated in the organization of prestigious international conferences, such as IEEE Congress on Evolutionary Computation (CEC) or the ACM Genetic and Evolutionary Computation Conference (GECCO).

Aitor Sanchoyerto is currently pursuing the Ph.D. degree in MC services management in 5G with UPV/EHU. He is also an Engineer with more than 20 years of experience in Telecommunications Projects in Public Safety Sector: Defense, National Security, and Transport. He is specialized in interoperability protocols between organizations with different levels of security and radio technology, whose objective is to be able to establish voice and data communications in a safe way. In recent years, he has been actively working on convergence projects from narrow-band radio communications to broadband communications based on the 3GPP-defined standard for mission-critical services.

Luis Zabala received the B.S. and M.S. degrees in telecommunications engineering from the University of the Basque Country, Spain, in 1998, where he is currently pursuing the Ph.D. degree. Prior to joining the University of the Basque Country in 2008, he worked for Gamesa Corporation, where he had 7 years of network engineering experience. He is currently a Lecturer with the University of the Basque Country. He has been cooperating in different R&D projects related to modeling of network monitoring systems and projects about telecommunications QoS evaluation and measurement. He conducts research in the areas of modeling and performance evaluation in high-speed networks and network function virtualization.

Bego Blanco received the B.S., M.S., and Ph.D. degrees in telecommunications engineering from the University of the Basque Country, Spain, in 2000 and 2014. She currently works as a Lecturer and a Researcher with the Faculty of Engineering in Bilbao. Her research interests include PQoS/QoE/QoS assessment and multicriteria optimization in 5G networks.

Fidel Liberal received the Ph.D. from the University of the Basque Country (UPV/EHU) in 2005. He works in the UPV/EHU, where he leads different international 5G and mission critical communications related R&D projects. His current research interests include efficient service deployment over 5G, particularly in mission critical scenarios. He has coauthored more than 90 international journal and conference articles in different telecommunication areas, most of them on broadband mobile networks. He is a well-recognized expert in the 5G and Mission Critical communications environment. In recent years, he has been very active in the field of Public Safety communications. As a result of an intense dissemination activity, he is connected with some of the most relevant stakeholders in the sector, such as EENA, PSCE, TCCA, NIST-PSCR, and standardization bodies, such as ETSI, 3GPP and ITU-T. He has also Co-Founded two security and mission critical communications related Spin-Offs.