A Q-learning strategy for federation of 5G services

Kiril Antevski*, Jorge Martín-Pérez*, Andres Garcia-Saavedra†, Carlos J. Bernardos*, Xi Li†,
Jorge Baranda‡, Josep Mangues-Bafalluy‡, Ricardo Martnez‡, Luca Vettori‡
*University Carlos III of Madrid, Spain, †NEC Laboratories Europe GmbH, Germany,
‡Centre Tecnològic de Telecomunicacions de Catalunya (CTTC/CERCA), Spain

***Abstract*—5G networks aim to provide orchestration of services across multiple administrative domains through the concept of federation. In this paper, we are exploring the federation feature of a platform for 5G transport network of vertical services. Then we formulate the decision problem that directly impacts the revenue of 5G administrative domains, and we propose as solution a Q-learning algorithm. The simulation results show near optimum profit maximization and a well-trained Q-learning algorithm can outperform the intuitive "greedy" approach in a realistic scenario.**

***Index Terms*—multi-domain, federation, NFV, algorithms, machine-learning**

I. Introduction

The 5G technology is evolving the mobile networks towards new design paradigms of more complex, diverse and dynamic network services designed for specific vertical industries (e.g., Automotive, eHealth, Industry 4.0, etc.) referred as vertical services. The 5G-TRANSFORMER project [1] proposes flexible transport of vertical services using the introduced 5G-TRANSFORMER (5GT) platform.

Through the technologies such as the Network Function Virtualization (NFV) and Network Slicing, range of specific requirements are satisfied for each vertical industry. Vertical services are translated into network services (NFV-NS) containing all requirements and instructions to be deployed over underlying network and computational infrastructure. The dynamic slicing enables the optimal resource allocation that reduces the cost of the mobile operators. The 5GT platform leverages the network slicing to efficiently orchestrate different NFV network services (NFV-NSs) over single or multiple domains.

The concept of service orchestration across multiple administrative domains [2] is one of the key features of the 5GT platform referred as *federation*. The federation enables each 5GT administrative domain (e.g., mobile operator) to provide broader spectrum of services to the vertical customers with low-cost access to infrastructure capacity and global service coverage in external domains. Through the use of optimized orchestration strategies (e.g., Machine Learning, Artificial Intelligence, ...) [3]–[5], operators may significantly increase their profit.

The main goal of the paper is to (*i*) present the federation functionality provided by the 5GT framework; (*ii*) formulate a service deployment decision problem with an aim to maximize the administration domain's revenue; (*iii*) and to propose a solution to the decision problem, evaluated through simulation of federation scenarios.

The rest of the paper is organized as follows. Section II presents the 5G-TRANSFORMER baseline architecture with an accent on the federation functionality. Section III defines the formulation of the deployment decision problem, whereas in Section IV we present the solution through simulation experiments. Section V concludes the paper with remarks for future work.

II. Service and Resource Federation in 5GT

This section introduces the high-level architecture of the 5G-TRANSFORMER system, with a focus of the federation as an essential feature of the 5GT framework. The federation feature allows 5GT service providers to deploy and manage services in external domains through service federation and to obtain and control computational/network resources from other domains through resource federation.

A. Baseline 5GT architecture

The 5GT architecture is built with the goal of providing a platform with flexible and dynamic management features to serve the needs of multiple and heterogeneous services coming from different vertical industries. Such services can be concurrently instantiated over a shared infrastructure that combines multiple heterogeneous types of resources in terms of computing, storage and networking. The 5GT architecture consists of four main building blocks, namely vertical slicer (5GT-VS), service orchestrator (5GT-SO), mobile transport and computing platform (5GT-MTP), and monitoring platform (5GT-MON). Two domains are represented in Fig.1, each having a full stack consisting of these four blocks (5GT-MON not represented for the sake of simplicity).

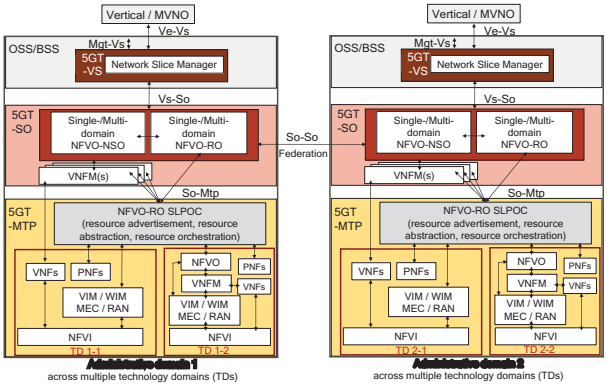

Fig. 1: 5GT Architecture

978-1-7281-5089-5/20/$31.00 ©2020 IEEE

The 5GT-VS is the entry point for vertical industries to access the 5GT platform. It provides a web portal and an API for (vertical) end-users, simplifying the process of requesting vertical services. The 5GT-VS exposes a catalogue of vertical services offered to vertical end-users, which are customized by the vertical users by setting parameters to mach their service requirements. The 5GT-VS translates from business-oriented vertical service requests into slice requests, which are eventually mapped to NFV-NSs. In turn, these NFV-NSs are requested by the 5GT-VS to the 5GT-SO.

The 5GT-SO manages the E2E orchestration and the lifecycle of the NFV-NSs. They are deployed by matching their requirements with the resource availability in the 5GT-MTP [6] through placement algorithms. The 5GT-SO contains two types of orchestrators: (*i*) the network service orchestrator (NFVO-NSO) and (*ii*) the resource orchestrator (NFVO-RO). Both orchestrators embody the functionalities of a typical NFV orchestrator (NFVO) [7]. Using the 5GT architecture, NFV-NSs can be deployed over single or multiple administrative domains (AD). For deployment over single (local) domain the 5GT-SO uses the So-Mtp interface towards the local domain 5GT-MTP. For multi-domain deployment and management, referred to as federation, the 5GT-SO establishes peer-to-peer interconnections with external ADs via the So-So interface [8] [9].

The 5G-MTP is responsible for orchestrating heterogeneous resources (computing, network, storage) available in each domain, exposing them to the 5GT-SO and serving its resource allocation requests to instantiate VNFs and to manage the connections of the underlying transport network. The 5G-MTP embeds plug-ins for interaction with the managers of each subset of resources : for (*i*) compute resources, the the Virtual Infrastructure Manager (VIM) or for (*ii*) inter-PoP networking connectivity, the WAN Infrastructure Manager (WIM). The 5GT-MTP generates an abstracted resource view towards the 5GT-SO via the So-Mtp interface according to policies in place.

The instantiation of NFV-NS follows a top-down workflow from the 5GT-VS down to the 5GT-SO and to the 5GT-MTP and underlying physical infrastructure. Note that a NFV-NS can be a single compact NFV-NS (containing multiple VNFs) or a modular composition of multiple (nested) NFV-NSs, referred to as composite end-to-end NFV-NS.

In the case of single domain instantiation, the important decision point is the placement of VNFs over the available resources. The Placement Algorithm, part of the 5GT-SO, generates placement decision for all the VNFs that composed the NFV-NS being instantiated [10]. The decision is based on the abstracted resource view provided by the 5GT-MTP and the NFV-NS requirements coming in the instantiation request from the 5GT-VS. In case of a federation scenario (multiple ADs), the number of deployment options significantly increases.

B. Service federation

In the 5GT framework, service federation applies to composite NFV-NS. A request for instantiation of a composite NFV-NS is sent from the 5GT-VS to the 5GT-SO. Each nested NFV-NS (of the composite) has a specific NFV-NS descriptor stored in the 5GT-SO repository which contains the service information for the specific nested NFV-NS. This information contains the number of VNFs, the computational resources used (i.e., CPU, memory, storage), the VNFs topology and requirements (i.e., bandwidth, latency requirements, service access points, etc.). The 5GT-SO decides for each nested NFV-NS whether to instantiate it locally (using the local 5GT-MTP) or to request service federation from a provider domain 5GT-SO via the So-So interface.

Service federation is the overall process of deploying nested NFV-NSs in a peering AD and stitching them (local and remote) to make an E2E composite NFV-NS. The AD that requests services is referred to as consumer domain while the AD capable of providing services is a provider domain. A requirement of federation scenarios is that domains must have business/service level agreements, in place. In turn, there are implicit technical implications, like setting peer-to-peer interconnection among themselves and exchanging their respective NFV-NS catalogues. Eventually, federation agreements should maximize the ADs' profit by extending the local domain service offering and by avoiding the rejection of NFV-NS deployment requests by requesting services to other domains, hence increasing computing/networking resource availability and service footprint.

The scope of our algorithm is to make fast decision for each nested NFV-NS whether to deploy it locally or to federate it in an external domain while maximizing the profit outcome. The decision point should be executed in the 5GT-SO prior deployment of a nested NFV-NS.

C. Resource federation

There is another federation scenario in which our algorithm can be used, that is, resource federation. Eventually, local domain resources of the 5GT framework may fail or they may be exhausted by running services. In this case, the 5GT-SO uses resource federation to request resources from a provider domain and to be able to use them as if they were local.

Resource federation is the overall process of consuming/providing computing and network resources from/to external federated domains. In this case, the decision point is also placed in the 5GT-SO module. After that decision, a consumer 5GT-SO requests resources to a provider 5GT-SO. Once granted, the consumer 5GT-SO takes control over provider's resources. Based on the agreed terms and conditions, the consumer domain controls and consumes the provided resources while the provider domain charges for their usage.

It is assumed that all agreements are already set and inter-domain interactions, such as advertising, requesting, offering and usage of available resources are working accordingly. By applying the resource federation perspective, our algorithm decides if resources, associated to a given service request, are requested to a federated domain. By so doing, long holds or repeated checking for available resources are avoided in case of resource scarcity or infrastructure failures.

III. Problem statement

Service federation and resource federation provide the 5GT service providers with various deployment options. In order to increase the revenue and to avoid resource shortage, it is important that for each deployment and/or scaling of NFV-NS, the 5GT-SO generates a profitable decision without significant increase of processing and re-calculation for available resources.

In section III-A, we formulate a Reinforcement learning (RL) based decision problem of the 5GT framework, that requires a solution to generate straight-forward deployment decision for each nested NFV-NS, and scale-up request.

Section III-B presents the intuitive "greedy" approach to generate a decision that maximizes the profit, and section III-C reformulates the RL-based decision algorithm as an optimization problem to maximize the profit.

Throughout this section we do not consider networking resources, nor a shortage of federated (external) resources. Although they are easily absorbed by the formulation, we decided not to include them to ease the problem readability. For simplification, we use network service to refer to NFV-NS.

A. Problem description

Let us consider a time-slotted system $t := \{1, 2, \dots, \}$. At the beginning of each slot, (i) we may receive one request to deploy a network service (or segment of a network service); or (ii) services already deployed may leave the system. A service request may arrive at instant t asking for $c^{(t)}$ CPUs, $m^{(t)}$ memory, and $d^{(t)}$ disk. And by that time the local domain will have $C^{(t)}, M^{(t)}, D^{(t)}$ CPU, memory, and disk, respectively.

The system state is represented as a vector $s^{(t)} = \left(c^{(t)}, m^{(t)}, d^{(t)}, C^{(t)}, M^{(t)}, D^{(t)}\right)$. An agent, in the local domain (i.e., the 5GT-SO itself), takes an action $a^{(t)}$ upon a service arrival at t. The agent can choose whether to consume its own resources for the incoming service ($a^{(t)} = 0$), to ask another domain to deploy it ($a^{(t)} = 1$), or to reject the service ($a^{(t)} = 2$).

The chosen action affects the instant reward $r^{(t)}$ that the local domain receives. An instant reward is the economic profit which the administrative domain receives per service deployment (locally or in federated domain). Indeed, in our system, the instant reward is determined by the state-action pair, i.e., $r^{(t)} = r(s^{(t)}, a^{(t)})$. And as long as the service arriving at t can be deployed locally, our system satisfies

$$r(s^{(t)}, 0) \geq r(s^{(t)}, 1) \geq r(s^{(t)}, 2) \tag{1}$$

We assume that the system state $s^{(t)}$ follows a distribution E, unknown *apriori*. Hence, the goal is to find an adequate policy $\pi(\mathrm{s}) : \mathcal{S} \rightarrow \mathcal{A}$ that maps a system state to an action. The above is modeled with an Markov Decision Process (MDP) with transition probability $p(s^{(t+1)} \mid s^{(t)}, a^{(t)})$, and the agent uses a policy π [11] to give a trajectory of states, actions and rewards $h_{1:T} := (s_1, a_1, r_1, \dots, s_T, a_T, r_T)$ over $\mathcal{S} \times \mathcal{A} \times \mathbb{R}$. The return from a state at time t is defined as the sum of discounted future rewards

$$R(t) = \sum_{i=t}^{T} \gamma^{(i-T)} r(s^{(i)}, a^{(i)})$$

with discounted factor $\gamma \in [0, 1]$. This is a classic reinforcement learning (RL) problem and the agent's task is to *learn* a policy that maximizes the expected return from the start distribution $J = \mathbb{E}_{r^{(i)}, s^{(i)} \sim E, a^{(i)} \sim \pi\}}[R^{(1)}]$.

It is convenient for RL problems to describe an action-value function describing the expected return after taking an action $a^{(t)}$ given a state $^{(t)}$ (i.e., following policy π) as

$$Q^{\pi}(s^{(t)}, a^{(t)}) = \mathbb{E}_{r^{(i \geq t)}, s^{(i \geq t)} \sim E, a^{(i \geq t)} \sim \pi\}}[R^{(t)} \mid s^{(t)}, a^{(t)}]$$

and particularly, its recursive representation (Bellman equation):

$$Q^{\pi}(s^{(t)}, a^{(t)}) = \tag{2}$$
$$\mathbb{E}_{r^{(t)}, s^{(t+1)} \sim E}\left[r(s^{(t)}, a^{(t)}) + \gamma \mathbb{E}_{a^{(t+1)} \sim \pi}[Q^{\pi}(s^{(t+1)}, a^{(t+1)})]\right]$$

Since the expectation depends only on E, the agent can learn Q^{μ} off-policy, using transitions which are generated from a different stochastic process like in Q-learning [12].

B. A greedy approach

For comparison, we introduce a straightforward "greedy" approach to solve the decision problem by locally deploying an incoming service, as long as there are enough local resources. We latter (see section IV) refer this algorithm as the *checker* solution, since it checks the availability of local resources before deciding to federate, or locally deploy a service.

According to the RL problem defined in section III-A, this translates into:

$$a^{(t)} = \begin{cases} 0, & c^{(t)} \leq C^{(t)} \wedge m^{(t)} \leq M^{(t)} \wedge d^{(t)} \leq D^{(t)} \\ 1, & \text{otherwise} \end{cases} \tag{3}$$

This approach never rejects a service request, as it assumes that other administrative domains can always host the service. Thus, the instant reward $r^{(t)}$ for federating a service is below the one obtained with a local deployment (as stated in (1)), since it is considered that the consumer domain is paying a hosting fee to the provider domain.

C. Optimization formulation

To check the goodness of a RL solution, we reformulate the problem in section III-A as an optimization problem.

We use binary variable $a_i^{(t)} = 1$ to abbreviate $a^{(t)} = i$, $i \in 0, 1, 2$. Note that $a_i^{(t)} = 0$ means $a^{(t)} \neq i$. Then, $C_f^{(t)}$, $M_f^{(t)}, D_f^{(t)}$ the CPU, memory and disk resources (respectively) freed at time t due to a service leaving. The instant when such leaving service arrived is denoted by $p^{(t)} \leq t$.

These variables help us to impose the resource conservation constraints:

$$C^{(t)} = C^{(t-1)} - a_0^{(t)} c^{(t-1)} + a_0^{\left(p^{(t-1)}\right)} C_f^{(t-1)} \tag{4}$$

$$M^{(t)} = M^{(t-1)} - a_0^{(t)} m^{(t-1)} + a_0^{\left(p^{(t-1)}\right)} M_f^{(t-1)} \quad (5)$$

$$D^{(t)} = D^{(t-1)} - a_0^{(t)} d^{(t-1)} + a_0^{\left(p^{(t-1)}\right)} D_f^{(t-1)} \quad (6)$$

which state that at time t available resources must consider deployed and freed resources at $t-1$. As well resources should always stay above zero

$$C^{(t)} \geq 0, M^{(t)} \geq 0, D^{(t)} \geq 0, \quad \forall t \quad (7)$$

and only one action is performed at each instant t

$$a_0^{(t)} + a_1^{(t)} + a_2^{(t)} = 1 \quad (8)$$

With constraints (4) - (8) the total reward ($r = \sum_t r^{(t)}$) is maximized taking as objective function:

$$\max \sum_t \sum_{i \in 0,1,2} r(s^{(t)}, a_i^{(t)}) a_i^{(t)} \quad (9)$$

IV. Algorithm and simulation results

This section evaluates the performance of a RL solution that solves the deployment decision problem in a federated environment. Section IV-A presents a simple Q-learning solution that solves the RL problem of section III-A.

Section IV-B presents the setup used to derive simulation results. Then, section IV-C begins showing how Q-learning approximates to the optimal solution with an adequate selection of parameters. Finally, end of section IV-C removes the unlimited federated resources assumption, and analyzes how it affects the Q-learning performance.

A. Q-learning algorithm

Based on the RL problem defined in section III-A, we derived a Q-learning algorithm that generates decision for each incoming network service deployment at instant t.

The algorithm (Algorithm 1) presents how the agent performs the deployment decision. It decides which action to make for every service request happenning in episode $[0, t_{\text{end}}]$. Then it repeats the whole episode EP times and returns the actions vector $\{a_t\}_0^{t_{\text{end}}}$ to which it has converged. In the beginning, the agent obtains information regarding the state of the system. The state reflects the available computational resources in the environment. Thus at the start, it is assumed that all resources in both (local and federated) domains are available with no network service running (i.e. $state = 0$ at time $t = 0$).

At some point, a new network service request may arrive. Once the service arrives in the system, the agent decides an action (e.g., local deployment, federation, rejection).

The well-known RL technique, the Q-learning algorithm, uses a state-action matrix Q_T to maximize the reward while iterates through states. Typically the states are the rows and the actions are the columns of the Q-learning matrix. In our work, there are only three actions (i.e., local deployment, federation and rejection), while the number of states depend of the system configuration (i.e., amount of computational resources). Algorithm 1 represents state $s^{(t)}$ as a vector of the available computational resources. Thus, Q_T has $\mathcal{O}(N^{dim(S)})$ entries with $N = \max_i s_i^{(t)}$ being the maximum capacity among all resources represented in the state vector, and $dim(S)$ the number of different resources considered (e.g., $dim(S) = 3$ if CPU, memory and disk are represented in the state vector).

Data: environment, EP
Result: $\{a_t\}_0^{t_{\text{end}}}$

1 $s^{(t)} \leftarrow \mathbf{0},\ e \leftarrow 0,\ r \leftarrow 0$;
2 $t_{\text{end}} \leftarrow$ environment.lastService();
3 **while** $e \leq EP$ **do**
4 $Q_T \leftarrow \mathbf{0}$;
5 $t \leftarrow 0$;
6 $r \leftarrow 0$;
7 **while** $t \leq t_{end}$ **do**
8 **if** $s^{(t)} == \emptyset$ **then**
9 $s^{(t)} \leftarrow$ environment.getState(t);
10 **end**
11 $a^{(t)} \leftarrow max_a \left\{Q_T[s^{(t)}, a]\right\} + \frac{1}{e+1}\text{unif}[0, 2]$;
12 environment $\leftarrow$ DeployService($s^{(t)}, a^{(t)}$);
13 $(s', r^{(t)}) \leftarrow$ environment.current(t);
14 $Q_T[s^{(t)}, a^{(t)}] \leftarrow (1-\alpha) Q_T[s^{(t)}, a^{(t)}] + \alpha \left(r^{(t)} + \gamma max_a Q_T[s', a]\right)$;
15 $s \leftarrow s'$;
16 $r \leftarrow r + r^{(t)}$;
17 $t \leftarrow t + 1$;
18 **end**
19 $e \leftarrow e + 1$;
20 **end**

Algorithm 1: Q-learning decision

At the beginning ($t = 0$), the Q-learning table is initialized to all zeros. Therefore the agent picks random action at the start, initiating the *learning* process.

Once a decision has been made, the deployment is executed locally, federated, or the service is rejected. The completed action *transits* the environment to a new state and generates instant reward (line 13). This means if the network service has been successfully deployed in the local domain, the agent can calculate the remaining available resources (i.e. the new state) and the immediate revenue of the deployment. However, if the agent decided for local deployment despite the lack of local resource, no transition to other state occurs, but a negative reward is generated for the performed action.

The instant reward and state transition enables the agent to *learn* (line 14) thanks to the Bellman equation (2). Additional fixed parameters represent the learning rate (α) and the discount factor (γ). The chosen values for the learning rate and the discount factor directly impact on the performance of the algorithm, this matter is evaluated in section IV-C. Afterwards the state of the system is updated with the new state and the instant revenue is added to the total revenue.

For each next incoming network service, all the steps of the algorithm are executed. The agent is continuously $learning$ and every next decision for an action is more knowledge-driven.

B. Simulation environment

We have set-up a simulation environment consisting of a local domain and external - federated domain. Both domains are capable of network service deployment that contains only computational resources (i.e. CPU, memory, disk). The local domain is configured to have limited amount of computational resources, whereas the federated domain is optionally set to have both limited and infinite computational resources.

Incoming requests for network service deployment are based on an Poisson arrival process of big B and small S services. The parameters for both big B and small S services are shown in Table I. The arrival process in the environment simulates 30 days of Poission arrivals of big and small services.

Each service request can be either *i)* deployed locally, *ii)* federated or *iii)* rejected. Whatever algorithm is used, the agent gets full reward(r_{big} or r_{small}) for local deployment; for federation, the consumer domain agent gets reward 1; and for rejection the reward is zero. In case the agent decides to locally deploy or federate a service over a full infrastructure, the agent receives penalty($-r_{big}$ or $-r_{small}$).

C. Performance evaluation

To evaluate the performance of the Q-learning algorithm described in IV-A, we performed set of simulation experiments.

First, the performance of the Q-learning algorithm depends on how the learning rate (α) and the discount factor (γ) are set up. To tune it and derive the best tuple (α, γ), we performed a set of simulations (≈ 400 simulations) exploring the combinations in the value range $[0, 1]$ with step 0.5 for each variable. Using the environment described in previous section IV-B, the local resources are finite (e.g. CPU=10; Memory=100GB, Storage=400GB), where the federated domain has unlimited resources. Fig. 2 shows the simulation results, presenting the total revenue that the algorithm produces for each tuple $(\alpha,\ \gamma)$. The x-axis presents the range of values chosen for the learning rate (α), the y-axis represent the discount factor (γ). The gradient color bar (on the right) present the indicator for the normalized reward revenue (i.e., obtained revenue r divided by the optimal policy π^* revenue r_{OPT}), ranging from low (*dark-red*) to high (*dark blue*). The results intelligibly show that the learning rate is the major contributor to profitable performance of the Q-learning algorithm. The learning rate's best performance is in the [0.35, 0.95] range, with negligible value range for the discount factor.

The performance evaluation for the Q-learning algorithm proceeds with 80-episodes simulation using the tuple values ($\alpha = 0.95$, $\gamma = 0.9$). Each episode runs the same arrival process of small and big services and the episode begins with all available local resources (CPU=10; Memory=100GB, Storage=400GB) and unrealistically infinite resources in the federated domain. I.e., the environment is reset to initial conditions at the start of each episode, while the Q-learning algorithm *learns* for the whole duration of the experiment.

TABLE I: Service arrivals

	Big B	Small S
Arrival rate	λ_B	$\lambda_S = 6\lambda_B$
CPU	10	1
Memory	20 GB	2 GB
Storage	1024 GB	20 GB
Life-time	5-10 days	1-4 days
Revenue	$r_{big} = 10r_{small}$	r_{small}

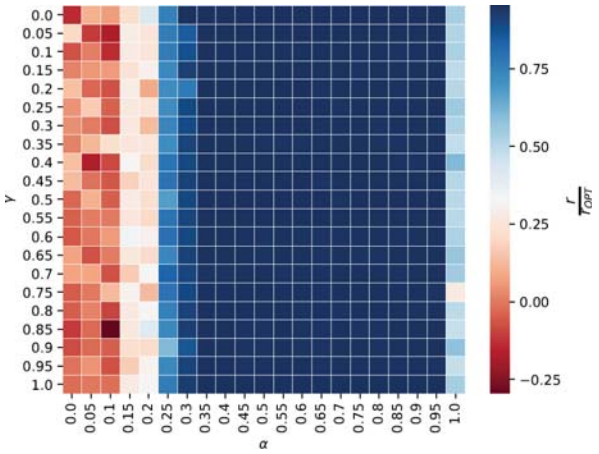

Fig. 2: Possible (α, γ) combinations under the assumption of unlimited federated resources.

The simulation results shown on Fig. 3, present all the episodes on the x-axis and the normalized revenue (per episode) on the y-axis. The solid blue curve presents the Q-learning reward evolution as the episodes increase. In the Fig. 3 we also introduced the results from three benchmark approaches:

- The orange (slash-slash) line represent the total reward of a single domain system, or if the federation option does not exist and there is no federated domain.
- The red (slash-dotted) line represent the total revenue of the "greedy" approach described in section III-B. Note that, here the agent first performs additional processing (e.g. scanning local resource capacity, checking databases, etc.,) before generating decision. If scan results are negative (insufficient resources), the agent federates the service in the federated domain. There are zero rejections in this case and the decision is made after the solution is checked and confirmed.
- The green (dotted) line represent the optimal solution generated from the optimization formulation of section III-C through the use of AMPL [13] and Gurobi [14].

Given a service arrival realization, results in Fig. 3 show that the Q-learning algorithm performs little over 95% of the optimal profit after $\approx$ 48 episodes. The "greedy" *checker* approach is slightly better ($< 1.5\%$) than the Q-learning algorithm. The "greedy" *checker* for each check adds more than *1.68 sec* [6] processing time to generate *a posteriori* decision. The added time corresponds to the size of the local domain resources that needs to be checked. On the contrary, the Q-learning generates *a priori* decision. Having a stable

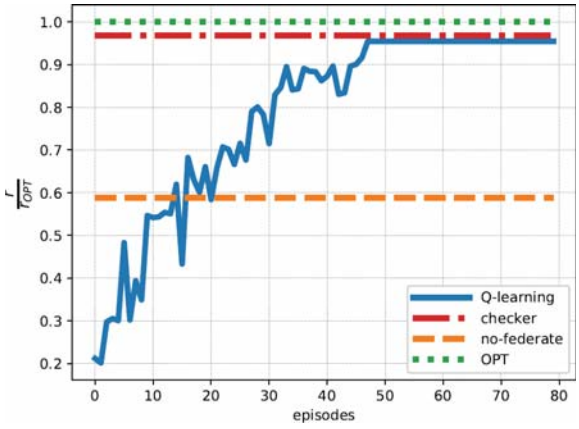

Fig. 3: Convergence of best (α, γ) combination under the assumption of unlimited federated resources.

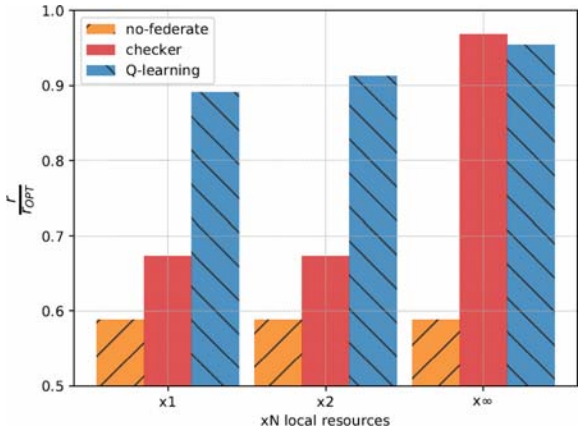

Fig. 4: Performance of Q-learning as federation resources increase. Training lapse of $EP = 200$ episodes.

scenario with infinite resources in the federated domain enables a near optimal solution for the "greedy" *checker*.

For more realistic evaluation of the Q-learning algorithm, we performed additional simulation experiments where the federation domain has finite resources. Therein the federation domain has been set to [*x1*, *x2*] times the local domain respectively. State vector $s^{(t)}$ now has three more components to represent the available CPU, memory and disk in the federated domain. Thus, $dim(S)$ duplicates, and the state space grows from $\mathcal{O}(N^3)$ to $\mathcal{O}(N^6)$, where $N = \max_i s_i^{(t)}$. Due to dynamicity of the scenario, the agent can not obtain precise status of the external resources. Therefore the "greedy" *checker* performs only local check and tries to federate a service request with no rejection. A penalty is applied when the agent tries to deploy over a full federated domain.

Fig. 4 compares the [*x1*, *x2*] scenarios to the infinite scenario of Fig. 3. Despite the growth in the state space, the Q-learning outperforms the "greedy" *checker* within 200 training episodes. Results imply that without the perfect knowledge in the "greedy" *checker*, the Q-learning can accommodate and *learn* the system dynamicity. Existence of penalty probability in the "greedy" *checker* is due to imprecise and highly complex operation to obtain information for all external resources.

V. Conclusion and Future Work

This work studies the performance of Q-learning to generate a profitable deployment decision in a federated ecosystem. Results show that Q-learning algorithms can provide the 5GT administrative domains near optimal earnings without additional time-consumption in an ideal scenario setup. In realistic scenarios, the Q-learning solution outperforms the "greedy" *checker* approach, with better computational efficiency. It is left, as future work, how to efficiently shrink the state space for a better problem scaling, and the study of alternative techniques as graph neural networks [15] to split incoming network service requests across local and federated domains.

Acknowledgment

This work has been partially funded by the EC H2020 5G-TRANSFORMER Project (grant no. 761536) and the EU H2020 5GROWTH Project (grant no. 856709).

References

[1] 5G-TRANSFORMER project. [Online]. Available: http://5g-transformer.eu/

[2] ETSI, "Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Report on architecture options to support multiple administrative domains," European Telecommunications Standards Institute (ETSI), Group Specification (GS) NFV 028 v3.1.1, January 2018.

[3] C. N. et al., "Machine learning aided orchestration in multi-tenant networks," in *2018 IEEE Photonics Society Summer Topical Meeting Series (SUM)*, July 2018, pp. 125–126.

[4] D. M. et al., "Artificial intelligence for elastic management and orchestration of 5g networks," *IEEE Wireless Communications*, pp. 1–8, 2019.

[5] Y. W. et al., "Network management and orchestration using artificial intelligence: Overview of etsi eni," *IEEE Communications Standards Magazine*, vol. 2, no. 4, pp. 58–65, December 2018.

[6] J. M. et al., "5g-transformer service orchestrator: design, implementation, and evaluation," in *2019 European Conference on Networks and Communications (EuCNC)*, June 2019, pp. 31–36.

[7] ETSI, "Network Functions Virtualisation (NFV); Management and Orchestration," European Telecommunications Standards Institute (ETSI), Group Specification (GS) NFV 001 v1.1.1, December 2014.

[8] L. Valcarenghi et al., "A framework for orchestration and federation of 5g services in a multi-domain scenario," in *Proceedings of the Workshop on Experimentation and Measurements in 5G*, ser. EM-5G'18. New York, NY, USA: ACM, 2018, pp. 19–24. [Online]. Available: http://doi.acm.org/10.1145/3286680.3286684

[9] X. L. et al., "Service orchestration and federation for verticals," in *2018 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, April 2018, pp. 260–265.

[10] K. A. et al., "Resource orchestration of 5g transport networks for vertical industries," in *2018 IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, Sep. 2018, pp. 158–163.

[11] R. Bellman, "Dynamic programming and stochastic control processes," *Information and Control*, vol. 1, no. 3, pp. 228 – 239, 1958. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0019995858800030

[12] C. J. C. H. Watkins, "Learning from delayed rewards," 1989.

[13] R. e. a. Fourer, "Ampl. a modeling language for mathematical programming," 1993.

[14] I. Gurobi Optimization, "Gurobi optimizer reference manual," *URL http://www. gurobi. com*, 2015.

[15] Z. W. et al., "A comprehensive survey on graph neural networks," 2019.