Scalable slice orchestration with DQN agents in beyond 5G networks

Pavlos Doanis[1], and Thrasyvoulos Spyropoulos[1, 2]
[1] EURECOM, France, first.last@eurecom.fr
[2] Technical University of Crete, Greece

***Abstract*—Beyond 5G systems are envisioned to host widely diverse services, with high Quality of Service requirements, over a common physical network infrastructure. Network slicing enables this vision by providing customized virtual networks on top of the physical one. These can be flexibly managed to fulfill stringent slice-specific service level agreements (SLAs) and ensure the efficient use of network resources, under time-varying resource demands. In this work, we examine the use of Reinforcement Learning (RL) for dynamic slice orchestration over a multi-domain setup, hosting multiple slices with end-to-end SLAs. Tabular RL algorithms can theoretically solve this problem optimally, even when the resource demand dynamics of slices are unknown. However, due to the combinatorial nature of the problem, the state and action complexity of these algorithms is prohibitive for realistic setups. To this end, we employ a Deep Q Network (DQN) to approximate the action-value function and therefore reduce complexity. However, since DQN is not suitable for problems with large action spaces, we propose a multi-agent DQN scheme to decompose the action into smaller components. We finally validate the proposed algorithm using realistic data, and show its scalability as well as the better performance of the derived policies compared to static baseline heuristics.**

Extended Abstract

Introduction

One of the key enablers for beyond 5G systems is network slicing, as it allows for flexible management of services, by creating customized virtual networks (”slices”) on top of the physical infrastructure. There is a twofold goal in optimal slicing: (i) to ensure some minimum performance requirements for each slice, specified by a corresponding SLA; (ii) to ensure efficient utilization of the limited network resources . The former is crucial, especially for services that cannot afford any performance degradation. The latter is also very important, since the more efficient utilization of the network resources, the more slices can be hosted with high reliability, and the higher the revenue for the network operator. From a different perspective, the less resources employed to reliably host a given number of services, the less the operating costs. [1]

A large part of the related literature focuses on solving the ”one shot” slice embedding optimization problem (i.e., based on average and/or static demands, without reconfiguration), by formulating it as a variant of the well studied virtual network embedding problem and using appropriate objectives and constraints [2]. Slices are commonly represented by graphs where the sets of nodes and links correspond to virtual network functions (VNFs) and virtual links (VLs) respectively. These must be mapped to physical nodes and links, while satisfying/optimizing the considered constraints/objectives. Many efforts have been made to tackle such problems [3], but a number of challenges still remain (both from modeling and algorithmic perspective). First, slices in beyond 5G networks span multiple technological domains, complicating the modeling of end-to-end KPIs (Key Performance Indicators) and increasing the problem’s complexity (due to the combinatorial nature of mapping VNFs and VLs to physical nodes and links). Second, the parameters that affect the performance of slices are often unknown and dynamically changing, which makes the use of ’learning’ algorithms imperative. [4], [5]

System Model

The system model consists of two main entities, the Physical Network and the slices on top of it.

Physical Network: it is represented by a weighted undirected graph comprising multiple domains. The nodes are servers or routers connected by physical links (all of which are characterized by some specified capacity).

Slices: they are represented by directed graphs, where the nodes are VNFs (processing tasks required by the slice) and the links dictate the order in which these VNFs must be traversed by a flow to receive the corresponding service. Each VNF and VL poses some demand for resources on the host server or physical link that is time-dependent according to the user generated traffic.

Configuration: it is the placement of VNFs to servers (control variables).

RL formulation

For the RL problem formulation, we consider that time is slotted and at each time-slot a central agent has to select an action based on the state of the system (the state summarizes all the important information needed for the agent to take an action). This action leads to a reward that is revealed when the system transitions to the next state (in the next time-slot). So, to formulate the RL problem of dynamic slice embedding we define the states, the actions, and the reward function of the system.

State: consists of the system configuration and the resource demands of all slices at time t.

Action: the configuration to be applied in the next time-slot.

Reward: it is the negative weighted sum of three different costs corresponding to: (i) reconfiguration (each VNF migration causes overhead and delays); (ii) node utilization (each active server has some operating cost); (iii) SLA violations (e.g. when the end-to-end delay for a slice exceeds the SLA).

RL algorithms

The combinatorial nature of placing VNFs to network servers leads to an exponential increase of the number of possible states and actions with the number of slices and available servers. Consequently, tabular algorithms like Q-learning, which maintain a table with the Q-values of all possible state-action pairs and learn each of them by explicitly visiting the corresponding state and taking the particular action multiple times, cannot be applied to realistically sized problems. For example, even in a toy scenario of a single domain network with 5 servers and 5 slices (consisting of only 1 VNF each with 5 possible discrete resource demand levels), would result to 5^{10} possible states and 5^5 possible actions (per state). The curse of dimensionality in our problem can be partially tackled by using a deep neural network (DNN) to approximate the Q function, instead of a table. This could decrease the number of the parameters to be learned by the agent and also exploit the DNN's generalization ability to learn the Q function without visiting all possible state-action pairs.

DQN is a widely used Q-learning algorithm that uses a DNN instead of the Q-table [6]. It is also equipped with some important additional features, like experience replay memory and a target network, for stability reasons. The input to the DNN is the state of the system and the output is the Q-values of all possible actions. However, DQN is suitable only for problems with a small action space, since the number of output neurons is equal to the number of possible actions, and an $\arg\max$ operation over all these actions is needed at each time-slot to choose the best of them.

Multi-agent DQN: the use of multiple DQN agents instead of one, where each agent is responsible for choosing the configuration of one slice or even the placement of a single VNF, is proposed to reduce both the state and the action complexity. So, considering one agent per VNF, the number of output neurons increases linearly with the number of nodes and the algorithm can be applied to large scale scenarios.

Simulation Results

We validate the multi-agent DQN algorithm by simulations using a physical network setup comprising 2 domains, with 6 and 3 servers respectively. We consider 10 slices on top of the physical network with various SLAs (e2e queuing delay constraints). Each slice consists of 2 VNFs, while the corresponding resource demands are imported from the Milano dataset [7] (each VNF maps to the traffic of a different base station). We use half of the 8926 available data points from the imported timeseries for the training of the DQN agents and the rest for testing. We compare the multi-agent DQN algorithm with two static baseline heuristics, called "group VNFs" and "split VNFs, as well as with a random actions policy. The "group VNFs" policy minimizes the node utilization cost, the "split VNFs" policy minimizes the SLA violation cost, while both of them have zero reconfiguration cost. Note that the tabular Q-learning and the single-agent DQN algorithms are not applicable to a scenario of that size (the number of possible actions is $\sim 4 \times 10^{12}$). The results are given in table I and demonstrate 42% lower cost for the multi-agent DQN policy compared to the best of the static baselines (split VNFs).

TABLE I
AVERAGE COST PER TIME-SLOT IN THE TESTING DATASET

multi-agent DQN	split VNFs	group VNFs	random
2.9	4.99	25.49	17.36

Conclusion and Future Work

In this work we discussed the scalability issues of centralized Q-learning algorithms for slice orchestration and proposed a multi-agent DQN scheme to reduce the state and action complexity of the problem. The results showcased both the scalability of the algorithm, compared to vanilla DQN and tabular Q-learning, as well as a significant cost improvement compared to static baseline policies.

Plans for future work include the examination of agent coordination as well as rollout Deep RL schemes in the fashion of the well known AlphaZero algorithm, which are known for their robust performance [8]. This type of algorithms can exploit a model of the system dynamics, as well as a given base policy that is known to perform decently, and further improve on that policy online using rollout. Another possible line of work could be to examine a fully distributed RL scheme to reduce the communication overhead in the system. Finally, it would be interesting to enhance the system model and include routing as one more control variable in the slice orchestration problem.

ACKNOWLEDGMENT

This work was supported by the SEMANTIC project, which has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 861165.

REFERENCES

[1] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Comms. Surveys Tutorials*, 2018.

[2] S. Vassilaras, L. Gkatzikis, N. Liakopoulos, I. N. Stiakogiannakis, M. Qi, L. Shi, L. Liu, M. Debbah, and G. S. Paschos, "The algorithmic aspects of network slicing," *IEEE Comms. Magazine*, 2017.

[3] F. Schardong, I. Nunes, and A. Schaeffer-Filho, "Nfv resource allocation: a systematic review and taxonomy of vnf forwarding graph embedding," *Computer Networks*, vol. 185, p. 107726, 2021.

[4] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "Aztec: Anticipatory capacity allocation for zero-touch network slicing," in *IEEE INFOCOM*, 2020.

[5] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "Deepcog: Cognitive network management in sliced 5g networks with deep learning," in *IEEE INFOCOM*, 2019.

[6] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[7] Telecom Italia, "Milano Grid," 2015.

[8] D. Bertsekas, "Lessons from alphazero for optimal, model predictive, and adaptive control," 2021. [Online]. Available: https://arxiv.org/abs/2108.10315