On Combining Reinforcement Learning and Monte Carlo for Dynamic Virtual Network Embedding

Ghina Dandachi, Yassine Hadjadj-Aoul, Abdelkader Outtagarts

On Combining Reinforcement Learning and Monte Carlo for Dynamic Virtual Network Embedding

Ghina Dandachi[1], Yassine Hadjadj-Aoul[1], and Abdelkader Outtagarts[2]

[1]*Inria, Univ Rennes, CNRS, IRISA, France*
[2]*Nokia-Bell Labs, France*

***Abstract*—Network slicing is one of the key building blocks in the evolution towards "zero touch networks". Indeed, this will allow 5G and beyond 5G networks to deploy services dynamically, on the same substrate network, regardless of their constraints. In this demo, we introduced a platform for dynamic virtual network embedding, a problem class known to be NP-hard. The proposed solution is based on a combination of a deep reinforcement learning strategy and a Monte Carlo (MC) approach. The idea here is to learn to generate, using a Deep Q-Network (DQN), a distribution of the placement solution, on which a MC-based search technique is applied. This makes the agent's exploration of the solution space more efficient.**

***Index Terms*—Virtual network embedding, Deep Q-Network, Control theory.**

I. Introduction

Networks' operators (NOs) have been focusing for several years on evolving their networks to accommodate the current and future services, with reducing operational and investment costs. Among these evolutions, the virtualization of network functions (VNFs) [1], and more generally services, is the one that has brought the most in-depth changes for network management. The concept of Network Slicing (NS) has been introduced in 5G to support a wide range of services within the same substrate network.

The problem of network slicing refers to the placement of constrained services [2]. This research problem generalizes the bin-packing problem, which has been extensively studied in the literature and well known to be an NP-hard problem. Indeed, service placement, as envisioned by 5G, implies not only the placement of nodes, representing the components of the service, but also links representing the network constraints existing between these different components (i.e., services in the form of a graph, namely Virtual Network Function Forwarding Graph (VNF-FG)).

Several approaches have been proposed in the literature to solve the service placement problem [3]. In [4], the authors proposed to combine a Deep Reinforcement Learning (DRL) strategy with a heuristic, to make the placement safer at the cost, however, of effectiveness. In [5], a Monte Carlo Tree Search (MCTS) strategy is proposed. It allows to find a sub-optimal solution to the placement problem, whereas the cost of a new research remains substantial since there is no learning. More recently, the authors, in [6], proposed to combine Graph Neural Networks with DRL to efficiently solve the service embedding problem.

Unlike the majority of existing works, we present, in this demo paper, a demonstrator of a DRL technique that considers dynamic requests of network slices with dynamic requirements and resource requests. The proposed solution leverages on DQN [7] with Monte Carlo (MC) search to learn the best placement solutions. The combination of these two approaches increases the exploration of the solutions' space. The proposed demo shows that our proposal improves the performance of heuristics such as a First-Fit, but also to improve the approach based on a pure MC strategy.

The rest of this paper is structured as follows. In Section II, we introduce the architecture of the proposed solution. On Section III, we introduce the experimentation settings and the obtained results. Section IV concludes the paper.

II. System Architecture

The proposed simulator demo is implemented in Python, using the PyTorch library. Figure 2 shows the GUI of the demonstrator, the user can choose the simulation time, the strategy, and the number of MC iterations. The problem resides in the ability of the controller to learn while having dynamic arrival of slices' requests, contrarily to most existing work, considering a fixed setting for slices' requests.

A. Architecture description

The architecture of our demo is presented in Figure 1. The Simulator loads the substrate information and receives dynamic virtual network requests (VNRs). The DRL agent extracts the features from the system state defined by the substrate available resources and VNR requirements. The MC strategy allows the explorer to generate several node-by-node DQN-based distributions for the VNR placement. The explorer selects a feasible placement with the highest revenue to the cost (r2c) and checks the nodes placement feasibility as well as edges connecting the VNR nodes. Then, it updates the substrate information with the new VNR placement. The DQN agent writes the placement solution to the batch memory for future learning.

1) System State:

- Substrate state: Defined in the file Substratesim.py to create the substrate instance and update its available resources.
- VNR state: Defined in vnrsim.py to create an instance of an arriving VNR with its requirements.

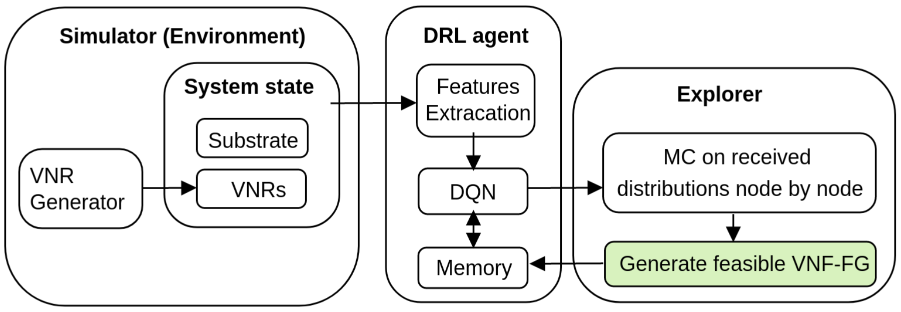

Fig. 1. Demo architecture

2) DRL agent: Defined in the file DQN.py.

- Two three-layers Neural Network (NN) extracts the system state features, for substrate and VNR, respectively.
- A DQN agent uses the state features in the three layers NN to provide the explorer with distributions node by node.
- Once the explorer generates a VNF-FG, the DRL agent updates the memory of the model for future learning.

3) Explorer: Defined in the file explorer.py

- Explorer_nodebynode:
    - receives the new VNR, deployed VNRs and, the substrate.
    - extracts the system state using the get_state function.
    - uses the DQN_MCselect_action function for nodes selection, verifies the nodes and edge mapping, then saves the VNF-FG chosen for this request, and updates the substrate.
    - sends information for the DQN agent to save it in the batch memory and learn from previous actions.
- get_state: Extract features from both substrate and VNRs in the form of Tensors and sends them to the DQN agent
- DQN_MCselect_action: This is the main function as the VNF-FG selection is decided in this step. Depending on the number of MC chosen, at each iteration, this function:
    - extracts substrate nodes selection probability distribution using DQN
    - selects a substrate node for the current VNR node
    - obtain a VNF-FG and calculates its corresponding r2c value
    - keeps track of all the states visited during the selected path for eventual usage by the DQN agent if the path is chosen to be the best path
- selects the VNF-FG with the highest r2c

4) Parameters: The Simulator parameters are defined in the file parameters.json and used during runtime. The main constants are:

- The simulation time, i.e, the number of episodes (SIM_TIME).
- The explorer strategy: First-Fit (FF), MC, or DQN (implicitly with MC if the number of iterations $N_{iter} > 1$).
- The number of MC iterations.
- The mean time between arrival for two VNR requests (MTBA).
- The minimum and maximum requested CPU (vcpu_range).
- The minimum and maximum request bandwidth (vbw_range).
- The maximum number of VNFs in a VNR (vnfs_range).
- The minimum and maximum VNR duration (vnr_duration_min, vnr_duration_max).

III. Experiments

All the modules of our demonstrator have been installed on a laptop. We consider for this demo the placement of VNRs using the following strategies: the First-Fit heuristic, the MC heuristic algorithm and the proposed DQN with MC solution. The First-Fit heuristic consists in placing the virtual nodes in the first available physical node, and then placing the paths using the Dijkstra algorithm.

To assess the performance of the proposed approach, we consider a network topology with 5 nodes and 4 full-duplex links. We considered the placement of dynamic requests of VNF-FGs, comprising between 3 VNFs each. For the physical nodes, we considered CPU in the interval $[20, 50]$. For physical links, we have considered bandwidth in the interval $[20, 50]$. For the virtual nodes, we considered CPU resources in the interval $[5, 10]$. For the virtual links we considered bandwidth in the interval $[5, 10]$.

Figure 3 shows the results obtained with different strategies: First-Fit (FF), DQN without MC (DQN), MC, and DQN with MC (DQN + MC) for different number of iterations (2 and 4 iterations). The obtained results show the revenue to the cost observed by the controller for a VNR placement defined as the amount of resources requested by a VNR divided by the actual amount of resources needed to achieve a VNR placement. We observe that DQN without MC achieves higher r2c than FF. By adding MC iterations to the path selection, the controller is able to reach higher values of r2c.

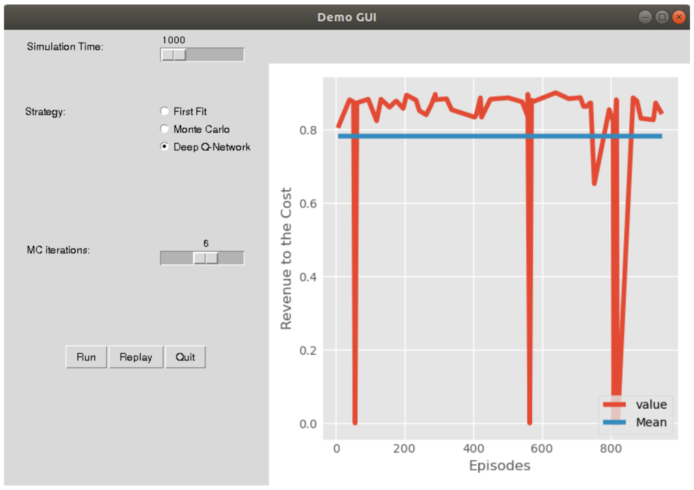

Fig. 2. Demo GUI

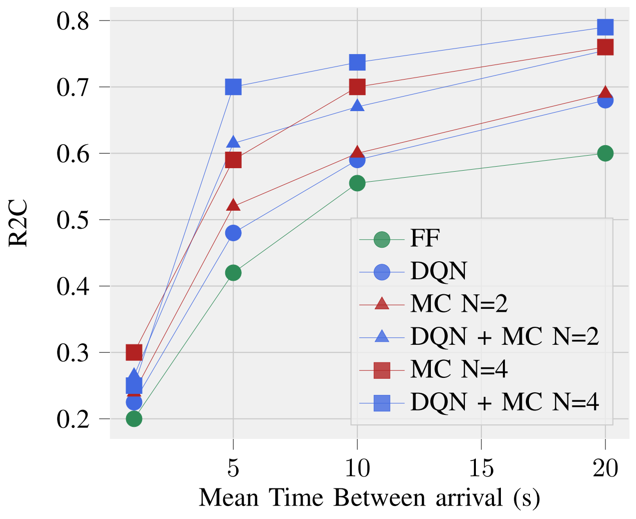

Fig. 3. Comparison of the r2c as a function of the Mean Time Between arrival for the three approaches FirstFit, Monte Carlo and DQN.

IV. Conclusion

Service placement is an important technology required by 5G networks, especially for Intent based orchestration that considers dynamic patterns as well as autonomous service placement and control. In this demo, we propose a dynamic slice placement tool for network slices based on deep reinforcement learning. It builds over the previously placed slicing, and offers a long term gain when compared to the baseline strategy (First-Fit). It also guarantees a minimum performance similar to that offered by First-Fit. In the future, we are planning to train our agent on a larger substrate network topology and larger VNRs. We also intend to extend our model definition and training algorithm to support the energy status of substrate nodes in order to detect nodes failure during embedding.

References

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

[2] P. Rost, C. Mannweiler, D. S. Michalopoulos, C. Sartori, V. Sciancalepore, N. Sastry, O. Holland, S. Tayade, B. Han, D. Bega, D. Aziz, and H. Bakker, "Network slicing to enable scalability and flexibility in 5g mobile networks," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 72–79, 2017.

[3] S. Redana, Bulakci, C. Mannweiler, L. Gallo, A. Kousaridas, D. Navrátil, A. Tzanakaki, J. Gutiérrez, H. Karl, P. Hasselmeyer, A. Gavras, S. Parker, and E. Mutafungwa, "5G PPP Architecture Working Group - View on 5G Architecture, Version 3.0," Jun. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.3265031

[4] P. T. A. Quang, Y. Hadjadj-Aoul, and A. Outtagarts, "A deep reinforcement learning approach for vnf forwarding graph embedding," *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1318–1331, 2019.

[5] S. Haeri and L. Trajković, "Virtual network embedding via monte carlo tree search," *IEEE Transactions on Cybernetics*, vol. 48, no. 2, pp. 510–521, 2018.

[6] Z. Yan, J. Ge, Y. Wu, L. Li, and T. Li, "Automatic virtual network embedding: A deep reinforcement learning approach with graph convolutional networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1040–1057, 2020.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.