Dynamic Machine Learning Algorithm Selection For Network Slicing in Beyond 5G Networks

Abdelmounaim Bouroudi, Abdelkader Outtagarts, Yassine Hadjadj-Aoul

Dynamic Machine Learning Algorithm Selection For Network Slicing in Beyond 5G Networks

Abdelmounaim Bouroudi
Nokia Bell Labs
Nokia Networks, France
abdelmounaim.bouroudi@nokia.com

Abdelkader Outtagarts
Nokia Bell Labs
Nokia Networks, France
abdelkader.outtagarts@nokia-bell-labs.com

Yassine Hadjadj-Aoul
INRIA, Univ Rennes, CNRS, IRISA
Rennes, France
Yassine.hadjadj-aoul@irisa.fr

***Abstract*—The advanced 5G and 6G mobile network generations offer new capabilities that enable the creation of multiple virtual network instances with distinct and stringent requirements. However, the coexistence of multiple network functions on top of a shared substrate network poses a resource allocation challenge known as the Virtual Network Embedding (VNE) problem. In recent years, this NP-hard problem has received increasing attention in the literature due to the growing need to optimize resources at the edge of the network, where computational and storage capabilities are limited. In this demo paper, we propose a solution to this problem, utilizing the Algorithm Selection (AS) paradigm. This selects the most optimal Deep Reinforcement Learning (DRL) algorithm from a portfolio of agents, in an offline manner, based on past performance. To evaluate our solution, we developed a simulation platform using the OMNeT++ framework, with an orchestration module containerized using Docker. The proposed solution shows good performance and outperforms standalone algorithms.**

***Index Terms*—Deep Reinforcement Learning; Algorithm Selection; Containerized Networks; Virtual Network Embedding.**

I. Introduction

Advanced 5G and 6G mobile networks will bring enhanced capabilities and performance, enabling various use cases for different industries and individuals [1]. This is largely due to virtualization technologies such as Network Function Virtualization (NFV) and Software-Defined Networks (SDN), which have been implemented across transport and core domains. NFV makes it possible to share physical infrastructure between multiple services by virtualizing the substrate network, while SDN separates the control plane from the data plane in the network elements, providing network operators with greater flexibility and programmability. Network Slicing has become a reality, thanks to these advancements.

The coexistence of heterogeneous network functions and services on the same substrate network results in a resource allocation problem known as the Virtual Network Embedding (VNE) problem. VNE has been proven to be an NP-hard optimization problem that has attracted a lot of attention, leading to numerous approaches being explored to solve it [2]. Among these approaches, Deep Reinforcement Learning-based solutions have shown promising results, with the ability to efficiently explore large problem instances [3].

In this paper, we present a new approach to solving the Virtual Network Embedding (VNE) problem based on the Algorithm Selection paradigm. This approach combines the strengths of different algorithms to solve a specific problem. We use a portfolio of state-of-the-art Deep Reinforcement Learning (DRL) based approaches for the VNE problem and perform a selection choice of one of the agents to solve a given VNE problem instance for a given scenario. We introduce a cooperative/competitive framework in which agents compete for better placement rewards while sharing experiences to enhance global learning efficiency. As a demo paper, we present the test-bed used to evaluate an Offline Algorithm Selection approach called ESBAS (Epochal Stochastic Bandit Algorithm Selection), first presented in [4]. In ESBAS, the execution time horizon is split into epochs of exponential size, during which DRL agents do not benefit from the acquired experience to update their policies. In our paper [5], we presented an adapted version of ESBAS for the VNE problem. We tested the Algorithm Selection approach using DRL agents with different strengths and performances to solve the VNE problem. Our approach showed improved performance and robustness compared to standalone algorithms, demonstrating its potential to address the challenges of dynamic network environments.

The rest of this paper is organized as follows. In the second section, we present the architecture of the test-bed used to evaluate the AS approach. In the third section, we introduce the experimentation settings, technical choices and the obtained results. Finally, in the fourth section, we conclude the paper.

II. Algorithm Selection Test-Bed Architecture

A. Overview

In Figure 1, we present the simulation test-bed used to evaluate different DRL-based approaches for solving the VNE problem. The test-bed is built following a microservice architecture and consists of two main components implemented in Python and C++, respectively. The first component, called the Orchestrator, acts as the control part of the platform and is responsible for making placement decisions for arriving Virtual Network Requests (VNR) using a DRL-based placement policy. The Orchestrator uses TensorFlow for DRL models creation and learning, and NetworkX for graph structure manipulation. The second component is the simulation platform, where the placement decisions made by the Orchestrator are executed. To achieve this, we used the OMNeT++ simulation

framework, which enables the simulation of a network topology with specific network resources, traffic generation, and Quality of Service (QoS) metrics retrieval. Both components communicate via a shared memory and are packaged in Docker containers.

To execute the simulation, we build a network model using the generated action from the shared memory and execute the simulation using the "NetworkRL" makefile. The simulation outputs are retrieved using the "Statistic" module and stored in the shared memory. On the other side, the "Environment" module collects the simulation outputs and calculates the reward based on QoS metrics (CPU, Storage, RAM and Bandwidth) returned. In the following, we provide an in-depth look at each component and its corresponding modules.

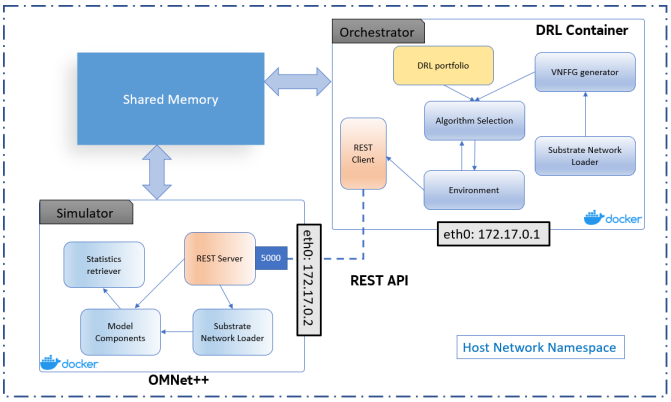

Fig. 1. DEMO TEST BED.

III. Architecture Description

The simulation process starts with loading the substrate network topology into the OMNeT++ simulator, as shown in Fig.1, using the "loadGml" function. This topology is used to simulate the substrate network. On the Orchestrator side, the initialization phase begins. Firstly, we load the created topology using the "Loader" module from the shared memory. Secondly, we generate a set of VNF-Forwarding Graph (VNF-FG) requests using the "generateRandomVNFFG" function. These VNG-FGs are intended to be placed on the substrate network using a placement strategy. Thirdly, we create a portfolio consisting of a set of DRL agents, among which the AS approach performs the selection.

Once the initialization phase is complete, the ESBAS algorithm starts by selecting an agent from the portfolio. The selected agent generates an action for a given VNF-FG placement request. The generated action is then sent to the "Environment" module to create a feasible placement action. The "Environment" module applies the Weighted First Fit Algorithm (W2FA), which is presented in [3]. Once a feasible placement action is created, it is sent to the simulator to be executed on the substrate network using the "runOmnet" method called via the REST API of the simulator.

The main components and modules of the proposed VNE solution are as follows:

1) Orchestrator:
   - Substrate Network Loader: This module is defined in the `Loader.py` file and is used to load a substrate network topology to the orchestrator.
   - VNFFG generator: This module is defined in the `Generate-vnffg.py` file and is used to generate random VNF-FGs with random values for each resource type.
   - DRL portfolio: This module is defined in the `main.py` file as a list of DRL agents to be selected and executed for an epoch by the Algorithm Selection module.
   - Algorithm Selection: This module is defined in the `main.py` file to perform the algorithm selection using the ESBAS selection approach.
   - Environment: This module is defined in the `Environment.py` file and implements a basic RL agent environment for the VNE problem (State, Action, Step, Reward).
   - REST Client: This module is defined in the `Rest_client.py` file and is used to exploit the Simulation platform API.
2) Simulation Platform:
   - REST Server: This module is defined in the `Rest_server.py` file and exposes the Simulator API which executes the orchestration actions and returns a status code.
   - Substrate Network Loader: This module is defined in the `Rest_server.py` file and is used to load the substrate network, create nodes, channels, and ports for an OMNet++ simulation instance.
   - Model Components: This module contains simple and compound modules definitions, used to simulate the substrate network along with the underlying running logic. The substrate network is described in the `NetworkAll.ned` file and the simulation parameters are specified in the `omnetpp.ini` file.
   - Statistic Retriever: This module is defined in `Statistic.cc` and is used to retrieve traffic delay and packet loss statistics. The collected statistics are shared through the shared memory with the orchestrator instance using text files. These files are then collected by the "Statistic Retriever" module in the orchestrator and used to calculate the reward and the new state.

IV. Experiments

The test-bed components are packaged in Docker containers and deployed on a single machine. We utilize the "Docker Bind Mounts" feature to share memory and the default Docker network for inter-container communication. To evaluate the effectiveness of our approach, we consider the "Bt-Europe" topology [6] as the substrate network, which comprises 24 nodes and 37 full-duplex links. For each iteration, we place 10 VNF-FGs generated using the "Erdos-Renyi" model [7].

To perform the selection, we use a portfolio of three different DRL algorithms with different learning potentials: **W2FA_DDPG** which is the basic DDPG algorithm combined with the W2FA heuristic, **EEDDPG** which is an enhanced version of DDPG and **EEDDPG Evolution** which uses an evolutionary operator to enhance the exploration capabilities of EEDDPG. More details about the last two agents can be found in [3]).

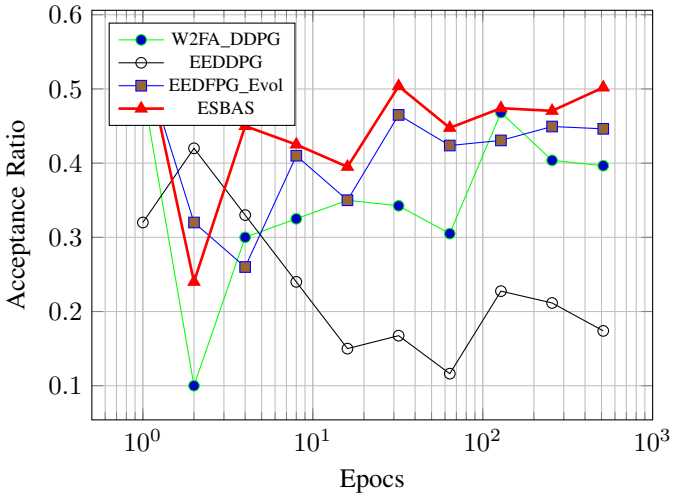

Fig. 2. Acceptance Ratio of ESBAS Compared with Standalone Algorithms.

Figure 2 shows the Acceptance Ratio (AR) of the ESBAS algorithm in comparison with the three agents. All algorithms are executed on exponentially increasing epochs, where the updates occur only at the beginning of each epoch. Among the three agents without ESBAS, EEDDPG Evolution proves to be the best agent in large epochs, which aligns with the findings of the authors in [3]. The success of achieving a high Acceptance Ratio early on in the learning process can be attributed to the capabilities of EEDDPG with Evolution, which enables rapid and effective learning. Note that in exponential epochs with less frequent updates, the other agents underperform compared to EEDDPG with Evolution. When compared with ESBAS, it is evident that ESBAS outperforms all the three standalone agents in all larger epochs starting from the third one.

In Fig. 3 we present the Selection Ratio made by ESBAS of three standalone agents. It is observed that in the first few epochs, ESBAS tends to make poor selection decisions. However, from the 5th epoch onward, the algorithm begins to consistently select W2FA_DDPG and EEDDPG with Evolution. This can be explained by the fact that with larger epochs, ESBAS has enough time to explore the different agent's potentials and make accurate selection decisions. As a result, we notice the elimination of both EEDDPG and W2FA_DDPG, while EEDDPG Evolution replaces them at over 80% of the selections in the final epoch, which explains the acceptance rate of ESBAS in Figure 2.

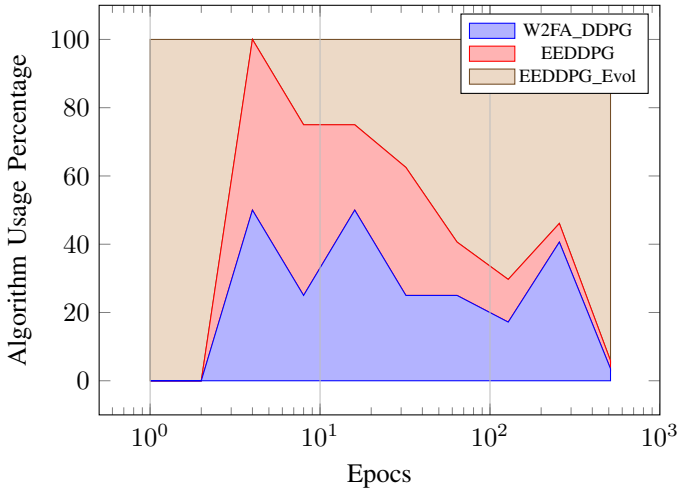

Fig. 3. Selection Ratio of ESBAS.

V. Conclusion

In this paper, we present a novel approach for addressing the challenge of network slicing in beyond 5G mobile networks, which involves meeting stringent and customized performance SLAs for different vertical services. Our proposed approach is based on the algorithm selection paradigm, and we have developed a demonstration tool to test and evaluate its potential in dealing with the VNE problem. The results of our experiments demonstrate the superiority of the Algorithm Selection solution over standalone DRL agents, highlighting the robustness of our approach in handling diverse scenarios and performance guarantees. Our future work will involve exploring an online selection version of the approach, where agents can update their policies frequently, as well as considering the use of dynamic arrival and departure of VNF-FGs instead of a fixed number of VNF-FGs deployed on the substrate network.

References

[1] Anutusha Dogra, Rakesh Kumar Jha, and Shubha Jain. "A survey on beyond 5G network with the advent of 6G: Architecture and emerging technologies". In: *IEEE Access* 9 (2020), pp. 67512–67547.

[2] Edoardo Amaldi et al. "On the computational complexity of the virtual network embedding problem". In: *Electronic Notes in Discrete Mathematics* 52 (2016), pp. 213–220.

[3] Pham Tran Anh Quang, Yassine Hadjadj-Aoul, and Abdelkader Outtagarts. "A deep reinforcement learning approach for VNF forwarding graph embedding". In: *IEEE Transactions on Network and Service Management* 16.4 (2019), pp. 1318–1331.

[4] Romain Laroche and Raphael Feraud. "Reinforcement learning algorithm selection". In: *arXiv preprint arXiv:1701.08810* (2017).

[5] Abdelmounaim Bouroudi, Abdelkader Outtagarts, and Yassine Hadjadj-Aoul. "Robust Deep Reinforcement Learning Algorithm for VNF-FG Embedding". In: *2022 IEEE 47th Conference on Local Computer Networks (LCN)*. 2022, pp. 351–354.

[6] Simon Knight. *The Internet Topology Zoo*. 2010. URL: http://www.topology-zoo.org/dataset.html.

[7] Paul Erdős, Alfréd Rényi, et al. "On the evolution of random graphs". In: *Publ. Math. Inst. Hung. Acad. Sci* 5.1 (1960), pp. 17–60.