Intelligent VNF Orchestration and Flow Scheduling

Nils Luca Rudminat

Summer term 2020

Abstract. Virtualizing network functions (e.g. firewall or router) is nowadays one concept to build a network. Instead of producing specialized hardware, they can be deployed on common servers which reduces in particular the arising cost for changing properties. However, this comes with a lot of new problems. One is the orchestration and the scheduling of traffic flows of several connected network functions on different servers. For a service provider a good and efficient solution is necessary to get a good revenue with less cost for his services.
This NP-hard problem can be solved exactly, heuristically or with the use of machine learning, in particular deep reinforcement learning (DRL). Exact and DRL solutions are slow, because of the complexity of the problem and heuristic algorithms are mostly deployed for specific cases, containing some assumptions. However, a DRL algorithm can be speed up by not just searching random in the action space, but by guiding the DRL agent with a good heuristic. This guidance can improve the convergence speed of the training by a factor of 23 while also improving the performance of the algorithm [2].

1 Introduction

Consider you are a service provider and you want to offer some service (e.g. internet) that consists of multiple functions (e.g. firewall, router, DPI...). You could just produce physical systems, one for every function, but this is neither flexibility or scalability nor saves your money. So you choose to virtualize them so that they run on common servers that can be rented from infrastructure providers. But now you are facing a few problems: What amount of a specific network function do you need? On which of the multiple possible servers from the infrastructure provider you install your network functions? If you choose to deploy more then one of a specific network function, to which of them you should schedule the incoming flow?

This are typical problems for a service provider and it is called the virtual network function (VNF) orchestration and flow scheduling problem. Orchestration is the installation and deinstallation of a VNF at a server and flow scheduling deals with the question to which of the activated VNF the traffic will be directed.
Luckily there are already solutions for this or strongly related problems. This NP-hard problem was solved mostly with Heuristicsor recently with deep reinforcement learning (DRL) (e.g [1]). However, heuristic solutions mostly simplify the model, take assumptions (e.g. they do not consider the end-to-end delay) or just solving the problem offline and recent DRL algorithms need a long time until the DRL agent is trained well if the action or the state space is large. Therefore Gu et al. [2] proposed a framework for DRL that let the training converge 23 times faster due to the guidance of the agent with a heuristic compared to the standard DRL approach.

2 Model

This explanation follows the formal model in [2].
In the VNF orchestration and flow scheduling problem servers from a infrastructure provider are given as an indirected graph. A server is represented as a node and a connection of two servers as a link

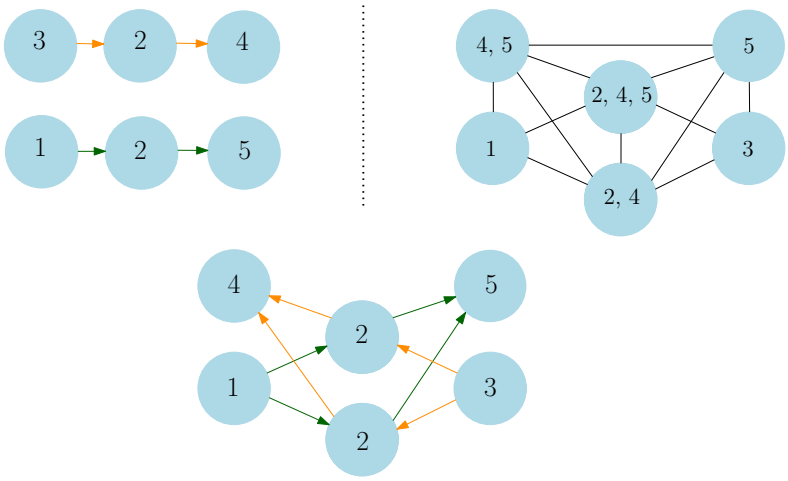

Figure 1: This example represents an installation of VNF at servers from an infrastructure provider at one timestep. The numbers represent the different network functions (namely 1,2,3,4 and 5).
On the top left are two SFCs that have to be installed on the servers from the infrastructure provider.
At the top right is the infrastructure graph with the servers and the links between them from the provider. The multiple numbers at the servers nodes represent the possible VNF that can be installed on the respective server.
On the button is one possible installation for the two SFCs. Note that a VNF can be shared by multiple SFC and there can be multiple of one VNF installed in the network. If the flow rate is low, you could also choose to install one of VNF 2. You could also choose to install two VNFs at one server from the infrastructure provider.

in the graph. At every server just a set of VNF can be installed, typically not all VNF can be installed on one server or prefer specific hardware.
Also set of service function chains (SFC) are given as a directed graph, where a node represents a network function. A SFC can be seen as a service in which different network functions must be executed in a predefined order. If a network function must be executed before an other network function it will be represented as an directed edge in the graph. The goal is to install VNFs on the servers of the infrastructure provider with low cost. An example of an installation can be seen in Figure 1.

The costs that are arising using a network from a infrastructure provider are put together as follows:

1. The setup cost (e.g. installing the virtual machine). If you let a VNF installed over multiple timesteps this cost just arises one time.

2. The cost for using a server, depending on e.g. the operation time and the cost per hour from the infrastructure provider for the specific server.

3. Communication cost for sending data between different servers. This cost depend on the amount of data sent.

4. End to end delay. This cost grows logarithmic with the delay of a flow.

The revenue is the money you get for the incoming flows. Hence, the utility is the revenue minus the costs, which has to be maximized. However, if the flow rate can not be predicted the utility can not be calculated beforehand. In this case the instant-utility which has to be minimized is defined as the summation of all costs without revenue and end to end delay.

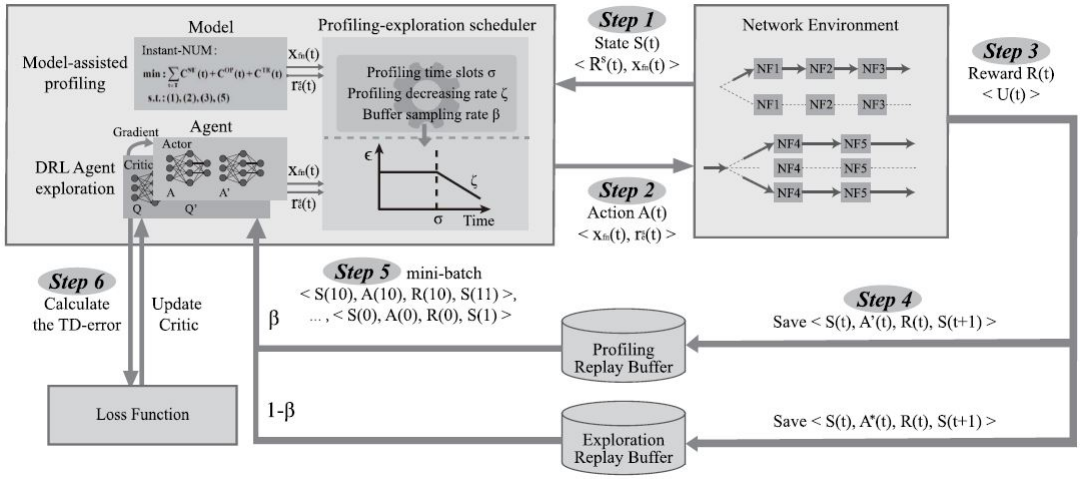

Figure 2: The DRL framework. It explains the different steps of the framework to optimize the DRL algorithm. Source: [2], Figure 1.

3 Model-Assisted DRL Framework

The core idea of [2] is that they use the deep deterministic policy gradient (DDPG) with the guidance of a heuristic. The DDPG algorithm needs as input tuples, containing the current state, an action, the reward and the next state, but instead of letting the agent explore the action space randomly at the beginning, they use a heuristic to create helpful tuples for the training.
In [2] the state is an vector containing the currently activated VNF at all servers and the flow rate between them. The action will be the modification of the flow rate and the activation status and the reward is the utility.
The model-assisted DRL Framework can now be summarized into the six steps in Figure 2:

1. At first the current state (activated VNFs and the current flow rate) must be observed from the environment.

2. Samples must be created to train the DRL-agent. A sample must contain the current state, an action, the reward for it and the next state you will get with the execution of the action. They propose two algorithms for picking the action based on the current observed state:
   Fist a bias-exploration agent where the agent explores the actionspace by himself which is just a slightly changed version of the normal exploration in DDPG.
   Second a profiling action generation algorithm which is creating the action based on a heuristic (based on the utility, if the flow rate can be predicted, or instant-utility if not).
   It is possible to only pick one algorithm. However, a better way according to the simulations is to use at the beginning the profiling action generation algorithm because the agent can learn better from the good actions of the heuristic than from his inexperienced choices. When the agent has got experience the exploration is used more so that the agent can also explore by himself. Because the profiling action generation algorithm can just generate actions based on the optimization in one timeslot, the exploration is necessary for learning particularly optimizations over multiple timesteps.

3. The reward for the action and the next state will be calculated.

4. Step 1-3 are repeated a few times for training until enough samples have been created. A sample is saved in the exploration and profiling replay buffer respectively.

5. The agent gets trained with a batch of samples

from the buffer.

6. The agent gets updated.

4 Implementation

The pseudocode for the framework is given in the paper, the rest (e.g. environment, solving ILP/LP problems ...) must be implemented without help of pseudocode [2]. There is no public implementation available.

5 Evaluation

Gu et al. [2] evaluate their approach with other DRL algorithms in particular also with their previous work [1]. The exact setup is also mentioned in the paper [2]. Roughly they used a real network topology with 43 nodes, embed 10 SFCs in it and use the instant-utility based heuristic for their algorithm. The SFC uses 30 different VNFs.
Their evaluation showed that their algorithm is faster then all of the other DRL algorithms compared with (23 times faster against the standard DRL approach and 19 times faster compared to their previous work). Though they need more time for one training step because of the calculation of the heuristic, the algorithm needs mush less episodes to converge to a good reward.
Furthermore they tested there algorithm with different variables (i.e. when to switch from the profiling action generation algorithm to the exploration algorithm). A too short or a too long use of the profiling action generation algorithm leads to a bad result. Already mentioned in chapter 3 the reason is that a short use of the algorithm leads to random actions in the beginning and a long use to overfitting because the instant-utility just optimizes for one timeslot and do not consider the actual end to end delay. With a good balance of the algorithms their approach leads to better results even after training. This is also the reason why the DRL algorithm with the heuristic based on the instant-utility performs a bit worse than the utility based heuristic, but the difference between them is not as huge as you would expect. Just using a heuristic solution (that do not consider end to end delay) achieves in their setup in the most cases the worst performance

6 Discussion

There are mainly 2 interesting questions to discuss. First: How does this work compares against other DRL algorithms in particular against their previous work [1]. It is quiet obvious that a DRL algorithm without guidance performs worse than the same DRL algorithm with guidance. However, their DRL algorithm also works better than their previous work. The main difference is that the predecessor for the algorithm just uses the bias-exploration method mentioned in chapter 3. So instead of having a profiling replay buffer they used a baseline buffer. Instead of putting tuples created from the instant-utility based algorithm into that buffer, they putting tuples from the bias-exploration that are better than the solution of instant-utility in it. This slight change together without the possibility to change the approach while the agent gets more experience leads to a worse result.

Second: Can you use the idea of the framework to improve other DRL algorithms even in other fields. The simple answer is yes you can. This framework could be build on top of other DRL-algorithms and also works with other problems. For other environments you need to deploy a heuristic for that specific problem and for other DRL algorithms you obviously need to implement the DRL algorithm, but how this framework behaves in this cases has not been researched yet and will be the most interesting question for the future.

References

[1] L. Gu, D. Zeng, W. Li, S. Guo, A. Zomaya, and H. Jin. Deep reinforcement learning based vnf management in geo-distributed edge computing. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 934–943, 2019.

[2] L. Gu, D. Zeng, W. Li, S. Guo, A. Y. Zomaya, and H. Jin. Intelligent vnf orchestration and flow scheduling via model-assisted deep reinforcement learning. *IEEE Journal on Selected Areas in Communications*, 38(2):279–291, 2020.