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. The scaling and updating of network functions is for example much easier and faster. 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 solutions and the training of DRL algorithms are slow because of the complexity of the problem and heuristic algorithms are mostly developed for specific cases, containing some assumptions. However, a DRL algorithm can be sped 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 compared to the standard DRL approach while also improving the performance of the DRL algorithm [2].

1 Introduction

Consider a service provider wants to offer some service (e.g. internet) that consists of multiple functions (e.g. firewall, router, DPI,...). The service provider could just produce physical systems, one for every function, but this is neither flexible or scalable nor saves money. So a virtualization of the network functions is necessary to run them on common servers that can be leased from infrastructure providers. But now a few problems arise that must be solved: What amount of a specific network function is necessary? On which of the multiple possible servers from the infrastructure provider should the network function be activated? If it was decided to activate more then one of a specific network function, to which of them should an incoming flow be scheduled?
That are typical problems for a service provider, called the virtual network function (VNF) orchestration and flow scheduling problem. Orchestration is the activation and deactivation of VNFs at the servers 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 and strongly related problems. This NP-hard problem was solved mostly with heuristics or 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 solve the problem offline and recent DRL algorithms take a great amount of 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 letting 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

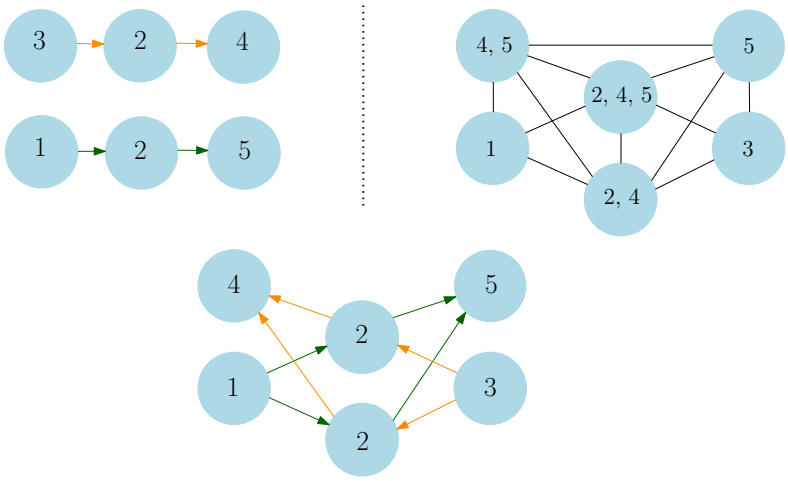

Figure 1: This example represents an activation of VNFs at servers from an infrastructure provider at one timestep. The numbers represent the different network functions (namely 1, 2, 3, 4 and 5).
On the top left are two SFCs that have to be installed on the servers from the infrastructure provider.
At the top right is the infrastructure graph with the servers, the deployed VNFs and the links between them from the provider.
On the button is one possible installation of the two SFCs. Note that a VNF can be shared by multiple SFC and there can be multiple of one network function deployed in the network. If the flow rate is low, it is also possible to activate just one of VNF 2 or to activate two VNFs at one server from the infrastructure provider. If the flow rate is high activating more VNFs is also possible (except for the start/origin function which has to be unambiguous).

problem servers from an infrastructure provider are given as an indirected graph. A server is represented as a node and a connection of two servers as a link in the graph.
Also set of service function chains (SFC) are given as a directed graph, in which 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 another network function, it is represented as a directed edge in the graph. The goal is to activate already deployed VNFs on the servers of the infrastructure provider with low cost. An example can be seen in Figure 1.

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

1. The setup cost (e.g. booting the virtual machine). If a VNF is activated 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 depends on the amount of data sent.
4. End to end delay. This cost grows logarithmic with the delay of a flow, because a certain amount of delay is more significant the less delay existed previously.

The revenue is the money received for the incoming flows. Hence, the utility is the revenue (the payment) minus the costs, which have 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.

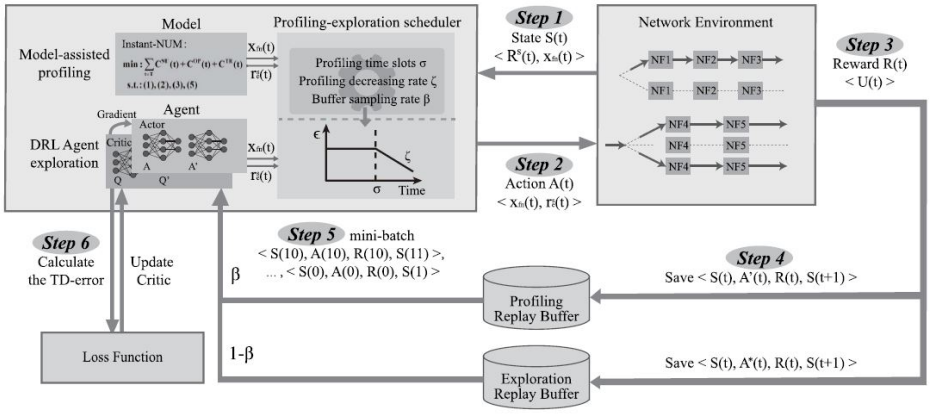

Figure 2: The DRL framework. It explains the different steps of the framework to optimize the DRL algorithm. Source: [2], Figure 1.

3 Model-Assisted DRL Framework

The core idea of [2] is to use the deep deterministic policy gradient (DDPG) with the guidance of a heuristic. The DDPG algorithm takes tuples as input, 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, a heuristic is used to create helpful tuples for the training.
In [2] the state is a vector containing the currently activated VNF at all servers and the flow rate between them. The action is the modification of the flow rate and the activation status and the reward is the utility defined in chapter 2.
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 have to be created to train the DRL-agent. A sample must contain the current state, an action, the reward and the next state. Gu et al. [2] propose two algorithms to pick the action based on the current observed state:
   First a bias-exploration agent which explores the action space by himself which is just a slightly changed version of the normal exploration in DDPG. The bias-exploration agent has a higher probability to explore edge cases (e.g. where the flow rate between VNFs is 0).
   Second a profiling action generation algorithm which creates 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 the profiling action generation algorithm at the beginning of the training because the agent can learn more effectively from the good actions of the heuristic than from its inexperienced choices. When the agent gained experience the exploration is used more, so that the agent can also explore by itself. 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 is calculated.

4. Step 1-3 are repeated 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. the environment, solving ILP/LP problems ...) has to be implemented without the help of pseudocode [2]. There is no public implementation available.

5 Evaluation

Gu et al. [2] evaluate their approach with other DRL algorithms, particularly 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 than 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 takes mush less episodes to converge to a good reward.
Furthermore, they tested their algorithm with different variables (i.e. when to switch from the profiling action generation algorithm to the exploration algorithm). A too short or 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 does not consider the actual end to end delay. With a good balance of the algorithms, their approach leads to better results even after training. Not considering the actual end to end delay is also the reason why the DRL algorithm with the heuristic based on the instant-utility performs a bit worse than the DRL algorithm with the utility based heuristic, but the difference between them is not as huge as expected. Just using a heuristic solution (that is not considering the end to end delay) without DRL achieves in their setup in most cases the worst performance.

6 Discussion

There are mainly two interesting questions to discuss.
First: How does this work compare against other DRL algorithms, in particular against their previous work [1]? It is quite obvious that a DRL algorithm without guidance performs worse than the same DRL algorithm with guidance. However, their DRL algorithm also works better than the one in 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 add tuples from the bias-exploration that are better than the solution of instant-utility. This slight change together without the possibility to change the approach while the agent gets more experience leads to a worse result.

Second: Can the idea of the framework be used to improve other DRL algorithms even in other fields? The simple answer is yes it is possible. This framework could be built on top of other DRL algorithms and also works with other problems. For other environments it is necessary to develop a heuristic for that specific problem and for other DRL algorithms it is obviously necessary to implement the DRL algorithm, but how this framework behaves in this cases has not been researched yet and is the most interesting question in 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.