Accelerating Network Slice Embedding with Reinforcement and Deep Reinforcement Learning

ABSTRACT

Network slicing plays an important role in next-generation communication systems (5G and beyond). With network slicing, mobile network operator is able to create various logical networks (slices) that are tailored to support specific services. One of the most challenging problem in network slicing is how to share the physical resources across multiple slices in an efficient way. This paper proposes some heuristics based on reinforcement and deep reinforcement learning to effectively solve this sharing problem. Extensive simulations have been performed to evaluate the proposed methods, in comparison with exact solutions obtained from solving an integer linear program. Numerical results demonstrate the effectiveness of the proposed learning-based algorithms in solving the problem of network slice embedding.

CCS CONCEPTS

• **Networks** → **Network resources allocation**.

KEYWORDS

network slicing, slice embedding, resource allocation, reinforcement learning, deep reinforcement learning, neural networks

ACM Reference Format:

. 2023. Accelerating Network Slice Embedding with Reinforcement and Deep Reinforcement Learning. In *Proceedings of The 18th Asian Internet Engineering Conference (AINTEC '23).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Network slicing is the creation of multiple virtual networks on a shared physical infrastructure, ensuring an agreed SLA for specific functionality. Each slice offers full network functionality. This is a paradigm shift from current implementations that makes network operations more efficient and cost-effective. Through automation, network slicing creates new revenue opportunities at scale to drive profitable growth [1, 5, 20].

A network slice is usually defined as a collection of Service Function Chains (SFCs) that are dedicated to support a specific service. Each SFC is a set of chained Virtual Network Functions (VNFs). These VNFs can be easily and flexibly initialized, launched, chained, and scaled to meet changing workload requests, requiring effective resource provisioning [5, 12].

AINTEC '23, December 12–14, 2023, Hanoi, Vietnam

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

In this paper, we tackle the problem of embedding network slices onto the physical network. This requires a series of decisions on the placement of VNFs and virtual links of each slice on the physical network, so as the resource demands of all these VNFs and virtual links are satisfied. The embedding should also fulfill various Service-Level Agreements (SLAs) such as end-to-end latency, coverage constraints, or the Quality of Service (QoS) associated with each type of services [6, 7].

Many approaches to solving this embedding problem have been studied. The exact method is to create a mathematical model in a form of Integer Linear Programming (ILP) or Mixed Integer Linear Programming (MILP) problem and optimize it using mathematical solvers/optimizers [10]. Although this produces the best results, the time required grows exponentially with the size of the problem. Alternative solutions include using heuristic algorithms which only provide sub-optimal solutions but can yield the solutions in a reasonable time.

In this paper, we primarily demonstrate the use of Reinforcement Learning (RL) and Deep Reinforcement Learning (DRL), specifically the Q-learning and Deep Q-learning algorithm as heuristic approaches to solving the problem of infrastructure network resource allocation for network slices, i.e., network slice embedding. Extensive simulations have been performed to compare the performance of these heuristic approaches to that obtained from the exact mathematical approach. Numerical results demonstrate that, while exact mathematical approaches are incapable of handling large problems, heuristic approaches can do it in a reasonable time.

The rest of the paper is organized as follows. Sec. 2 summarizes some related works. The mathematical model for the slice embedding problem is introduced in Sec. 3. Our RL and DRL algorithm to heuristically solve the problem is proposed in Sec. 4. The simulation and evaluation of the proposed approach are presented in Sec. 5. Finally, some conclusions and perspectives are drawn in Sec. 6.

2 RELATED WORK

Many work on using RL and DRL to solve the problem of slice embedding have been proposed in the literature. In what follows, we summarize the difference of these works, especially in how the RL environment is defined, i.e., the modelling of the state space, the action space, and the reward obtained by selecting an action from a given state.

Many different ways of defining action space have been introduced. For instance, in [16], an action involves selecting a candidate node from the physical network to perform the VNF placement. The same approach has been considered in [18], in which the actions encompass decisions about physical node embeddings. In [17] actions indicate whether a VNF can be deployed on a specific server node. The RL model in [19] involves selecting a Virtual Network Embedding (VNE) strategy as the action. The action space studied in [13] is the activation of new links between nodes. In [2, 3], actions involve placing VNFs within network slices. In [14], a binary

Table 1: Main notations.

Symbol	Description
$\mathcal{G}$	Graph reprensting the physical network
$\mathcal{N}$	Set of physical nodes in $\mathcal{G}$
$\mathcal{L}$	Set of physical links in $\mathcal{G}$
a_i	Available resources of physical node (i)
a_{ij}	Available resources of physical link ij
$\mathcal{G}_k$	Graph representing slice k
$\mathcal{N}_k$	Set of VNFs in slice k
$\mathcal{L}_k$	Set of virtual links in slice k
r_v	Resources requirements of VNF v
r_{vw}	Resources requirements of virtual link vw
$\mathcal{K}$	Set of slices k
π_k	Binary variable indicating whether slice k is admitted or not
$x_i^{v,k}$	Mapping indicator between VNF v and physical node i
$x_{ij}^{vw,k}$	Mapping indicator between virtual link vw and physical node ij

list selection of network nodes determines actions. These diverse action definitions reflect the unique tasks and decision spaces of each study.

The definition of the state spaces also varies in different studies. Some encompass both network and request states [2, 3, 17], while others focus on different attributes such as resource availability, node characteristics, and link attributes [13, 18, 19].

The calculation of rewards also diverges across studies. Certain studies employ explicit formulas considering parameters like revenue, cost, and throughput [16, 19], while others use predefined reward values for success or failure case, often associated with metrics like resource consumption, load balancing, and network improvement [2, 3, 17]. This diversity in reward calculation methodologies is indicative of the multifaceted objectives each study aims to achieve.

In addition, one can also find the difference in the way the objective is defined. These objectives include maximizing revenue [16], minimizing costs and maximizing throughput [18], jointly optimizing operation cost and total throughput [17], optimizing revenue-cost ratio [19], enhancing network score [13], and maximizing accepted VNF-Forwarding Graphs satisfying some pre-defined constraints [2, 3]. The specific goals guide the action selection and reward mechanisms, showcasing the unique directions taken by each study.

3 PROBLEM STATEMENT

3.1 Network Model

The physical network can be represented as a weighted directed graph denoted by $\mathcal{G} = (\mathcal{N}, \mathcal{L})$, where $\mathcal{N}$ is the set of physical nodes and $\mathcal{L}$ is the set of all available links within $\mathcal{G}$. Each physical node $i \in \mathcal{N}$ is characterized by its resource capacity a_i (e.g., computing, storage, or processing). Similarity, each physical link $ij \in \mathcal{L}$ has a resource capacity of a_{ij} (e.g., bandwidth).

Each network slice $k \in \mathcal{K}$ can also be represented as a weighted directed graph, denoted by $\mathcal{G}_k = (\mathcal{N}_k, \mathcal{L}_k)$. Here $\mathcal{N}_k$ represents the set of virtual nodes (or VNFs) and $\mathcal{L}_k$ represents the set of all virtual links within slice k. Each virtual node $v \in \mathcal{N}_k$ requires an amount r_v of physical resources. Similarly, each virtual link $vw \in \mathcal{L}_k$ between two VNFs v and w of slice k is characterized by a resource requirement of r_{vw}.

3.2 Variables

To formulate the slice embedding problem, we introduce the sets of variables $\boldsymbol{\pi} = \{\pi_k\}_{k\in\mathcal{K}}$ and $\boldsymbol{x} = \{x_i^{v,k}, x_{ij}^{vw,k}\}_{(i,ij)\in\mathcal{G},(v,vw)\in\mathcal{G}_k,k\in\mathcal{K}}$, where

- $\pi_k \in \{0,1\}$ is a binary variable indicating whether a slice k is admitted or not;
- $x_i^{v,k} \in \{0,1\}$ is the node mapping variable. $x_i^{v,k} = 1$ means that the virtual node v of slice k is mapped onto the physical node $i \in \mathcal{N}$, and $x_i^{v,k} = 0$ otherwise;
- $x_{ij}^{vw,k} \in \{0,1\}$ is the link mapping variable. $x_{ij}^{vw,k} = 1$ indicates that the virtual link vw of slice k is mapped onto the physical link $ij \in \mathcal{L}$, and $x_{ij}^{vw,k} = 0$ otherwise.

3.3 Constraints

An embedding solution can only be valid when it fulfills all of the following constraints.

Node capacity. The total resources that a given physical node $i \in \mathcal{N}$ allocated to VNFs of all slices should not exceed its capacity,

$$\sum_{k\in\mathcal{K}} \sum_{v\in\mathcal{N}_k} x_i^{v,k} r_v \le a_i, \quad \forall i \in \mathcal{N}. \tag{1}$$

Link capacity. Similar to constraint (1), the total resources that a given physical link $ij \in\in \mathcal{L}$ should not exceed the capacity of ij,

$$\sum_{k\in\mathcal{K}} \sum_{vw\in\mathcal{L}_k} x_{ij}^{vw,k} r_{vw} \le a_{ij}, \quad \forall ij \in \mathcal{L}. \tag{2}$$

Mapped only once. Each instance of a VNF $v \in \mathcal{N}_k$ can only be mapped to a single physical node $i \in \mathcal{N}$. This is to prevent service interruption or loss in case something goes wrong with the physical node, which would affect all the VNFs in the slice if they are running on the same physical node.

$$\sum_{v\in\mathcal{N}_k} x_i^{v,k} \le \pi_k, \quad \forall i \in \mathcal{N}, k \in \mathcal{K}. \tag{3}$$

Map all VNFs. This constraint ensures that, when a slice k is admitted (i.e., $\pi_k = 1$), all of its VNFs should be mapped onto the physical network,

$$\sum_{i\in\mathcal{N}} x_i^{v,k} = \pi_k, \quad \forall v \in \mathcal{N}_k, k \in \mathcal{K}. \tag{4}$$

Flow conservation. Finally, in order for the slice to operate as designed and meet its requirements, the order in which the VNFs are mapped onto the physical network must strictly follow the logic laid down by the slice,

$$\sum_{j\in\mathcal{N}} x_{ij}^{vw,k} - \sum_{j\in\mathcal{N}} x_{ji}^{vw,k} = x_i^{v,k} - x_i^{w,k}, \quad \forall ij \in \mathcal{L}, vw \in \mathcal{L}_k, k \in \mathcal{K}. \tag{5}$$

This constraint guarantees the continuous connection between each pair of physical nodes that are used to map the pair (v, w) of slice k.

3.4 Objective Function

Various objectives can be used for the slice embedding problem, e.g., minimizing embedding cost minimization, maximizing revenue for service providers, or reducing latency. This paper mainly focuses on the objective of maximizing the number of slices that can be

admitted to be embedded onto the physical network. This reflects the ability of the network to accommodate multiple service requests from different tenants while ensuring their quality of service. For this purpose, we define the number of admitted slices $N(\boldsymbol{\pi})$ as

$$N(\boldsymbol{\pi}) = \sum_{k \in \mathcal{K}} \pi_k. \tag{6}$$

Finally, the slice embedding problem (denoted as ILP-SE) can be cast as an ILP as follows,

$$\max_{\boldsymbol{\pi}, \boldsymbol{x}} N(\boldsymbol{\pi}), \quad \text{s.t. (1)–(5)}, \tag{ILP-SE}$$

which has been proven to be *NP*-hard [10] and thus becomes intractable when dealing with large-scale problem instances. To tackle this challenge, in what follows, we shall introduce a solution framework based on RL and DRL to solve the slice embedding problem at large scale.

4 RL AND DRL FRAMEWORK FOR NETWORK SLICE EMBEDDING

4.1 Definition of the RL Environment

We propose an RL/DRL-based framework to solve the slice embedding problem, in which all slices $k \in \mathcal{K}$ will be sequentially embedded onto the physical network $\mathcal{G}$. For each slice k to be embedded, each VNF and virtual link of k will also be embedded one-by-one onto $\mathcal{G}$.

The RL framework consists of three key elements: the state space $\mathcal{S}$, the action space $\mathcal{A}$, and the reward $\mathcal{R}$. For the embedding problem for a given slice $k \in \mathcal{K}$, we consider the following RL framework.

State. The state $\mathcal{S}_{k,t}$ represents the set of VNFs of slice k that have not yet been mapped at time t,

$$\mathcal{S}_{k,t} = \{v \in \mathcal{N}_k : \sum_{i \in \mathcal{N}} x_i^{v,k} = 0\}. \tag{7}$$

Each episode of the training process involves progressing from the initial state (comprising all VNFs) to transitioning through intermediate states (VNF mappings in sequence), until reaching a terminal state, when all VNFs of slice k have already been mapped.

Action. During the state transition process, the RL agent selects an action $\alpha \in \mathcal{A}_{k,t}$, where the action space $\mathcal{A}_{k,t}$ is the set of physical nodes that have not been assigned VNF mappings for slice k yet.

$$\mathcal{A}_{k,t} = \{i \in \mathcal{N} : \sum_{v \in \mathcal{N}_k} x_i^{v,k} = 0\}. \tag{8}$$

Reward. The process of selecting actions to transition between states is where the agent interacts with the environment. During this interaction, the environment provides feedback in the form of rewards, which signify the "quality" of the interaction process.

Since VNFs of a given slice k are sequentially mapped on to $\mathcal{G}$, we define the reward received from the selection of a physical node i to map the first VNF v of slice k as follows,

$$\mathcal{R}_{k,t} = M - \left(a_{i,t} - r_v\right), \tag{9}$$

where $a_{i,t}$ is the available resources of the physical node i at time t, and M is an arbitrate number to make the reward positive. This reward function favors the node resource utilization, i.e., selecting a physical node i that has sufficient resource to accommodate the requirement of the VNF v will be received a bigger reward than selecting a physical with many available resources.

For the remaining VNFs $v' \neq v$ of slice k, the reward for selecting physical node $i' \neq i$ is calculated as

$$\mathcal{R}_{k,t} = M - \left(a_{i',t} - r_{v'}\right) - \beta h_{i,i'}, \tag{10}$$

where β serves as a parameter that assesses the level of significance when traversing multiple hops and $h_{i,i'}$ represents the number of hops between i and i'. With this reward function, the algorithm tends to select the next physical node i' to map v' that is not far from the physical i that has been chosen to map VNF v, thus reduces the number of used physical links.

4.2 Q-learning Algorithm

Having the RL environment defined in Sec. 4.1, we introduce in this section a Q-learning-based algorithm to solve the slice embedding problem.

To perform the process of choosing an action efficiently, at each state $s \in \mathcal{S}$, we employ an ε-greedy [15] strategy as follows,

$$\alpha_t = \begin{cases} \text{argmax}_{\alpha \in \mathcal{A}_{k,t}} Q_t(\alpha), & \text{with probability } 1 - \varepsilon, \\ \text{random action}, & \text{with probability } \varepsilon. \end{cases} \tag{11}$$

This random selection of actions disorganises the experience correlation, thus solving the overfitting issue of neural networks.

To optimize the total reward, the Q-learning algorithm consistently acquires knowledge about the system by actively exploring it and making decisions iteratively. The choice of action in the current situation is influenced by the Q-value $Q(s_t, \alpha_t)$. These Q-values are stored in a Q-table. This learning procedure consistently modifies Q-values until the system can make optimal choices across all scenarios. The agent adjusts the action-value function using the Bellman equation [15] given by

$$Q(s_t, \alpha_t) \leftarrow (1 - \lambda)\, Q(s_t, \alpha_t) + \lambda \left(r_t + \gamma \max_{\alpha} Q(s_{t+1}, \alpha_{t+1})\right), \tag{12}$$

where r_t is the reward received at state s at time t when action α_t is performed; $Q(s_{t+1}, \alpha_{t+1})$ represents the Q-value of all subsequent actions α_{t+1} after performing action α_t at state s_t; γ is the discount factor used to weight the relationship between the current reward value and the future reward value; and λ is the learning rate. Based on the mechanism of update across multiple iterations, the agent adjusts the Q-value according to new experiences and received rewards. The algorithm stops when the Q-value is converged. We summarize in Algorithm 1 the procedure of Q-learning-based solution to solve the slice embedding problem (denoted as QL-SE).

4.3 Deep Q-learning Algorithm

The Q-learning algorithm is only suitable for relatively small scale problems. For larger problem instances, storing and processing Q-values in the Q-table leads to higher computing and storage resource consumption. To address this issue, DQL appears as a prominent solution. Instead of directly calculating Q-values and storing them in a Q-table, DQL employs a Deep Neural Network (DNN) to approximate Q-values corresponding to the input states and feasible actions. By training the DNN, we can determine the optimal action

Algorithm 1: QL Algorithm for Slice Embedding (QL-SE).

initialize: hyperparameters and environment;
foreach *slice* $k \in \mathcal{K}$ **do**
  foreach *episode* **do**
    Initialize initial state and action;
    while *not all states are explored* **do**
      Choose $s_t \in \mathcal{S}$;
      while *action satisfies constraints* **do**
        Select action w.r.t the ε-greedy strategy;
      Compute the reward, Q-value for the initial state, terminal state and other states: $\mathscr{R}_t = M - \left(a_{i',t} - r_{v'}\right) - \beta h_{i,i'}$; $Q(s_t, \alpha_t) \leftarrow (1-\lambda)\, Q(s_t, \alpha_t) + \lambda \left(r_t + \gamma \max_\alpha Q(s_{t+1}, \alpha_{t+1})\right)$;
    Update exploration rate: $\varepsilon = \varepsilon \times \varepsilon_{\text{decay}}$;
  Finding the optimal policy based on the Q-table;
  Update the available resources on $\mathcal{G}$;

without exploring all states, thus reducing the complexity of the network space.

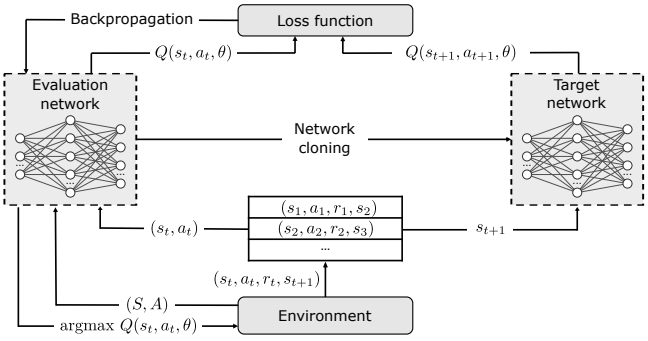

Figure 1: Architecture of the DQL-SE algorithm.

The architecture of DQL has two fundamental components, the target and the evaluation network, as depicted in Fig. 1. The evaluation network is used to approximate the Q-value $Q(s_t, \alpha_t)$ for the current state s_t. During the agent's interaction with the environment, accumulated experiences are stored in a memory buffer. The evaluation network then utilizes samples from this memory to update the weights and biases of the neural network through the gradient descent algorithm,

$$\theta := \theta - \lambda \frac{dL(\theta)}{d\theta}, \tag{13}$$

where θ represents the set of weights and biases of the neural network model, and λ is the learning rate that determines the rate at which the parameters are updated during the training process. This updating process adjusts the ability to approximate $Q(s_t, \alpha_t)$ as close as possible to $Q(s_{t+1}, \alpha_{t+1})$, thus achieving convergence.

After each specific training iteration, the weights and biases of the value network are cloned to the target network. The target network is used to predict $Q(s_{t+1}, \alpha_{t+1})$ for the next state s_{t+1}. Subsequently, with two Q-values $Q(s_t, \alpha_t)$ from the evaluation network and $Q(s_{t+1}, \alpha_{t+1})$ from the target network, the training process continues to minimize the following loss function $L(\theta)$,

$$L(\theta) = \frac{1}{2}\left(Q(s_{t+1}, \alpha_{t+1}, \theta^-) - Q(s_t, \alpha_t, \theta)\right)^2. \tag{14}$$

By minimizing the loss function $L(\theta)$, the interaction process helps adjust the value network to make the approximated $Q(s_t, \alpha_t)$ closer to $Q(s_{t+1}, \alpha_{t+1})$, thereby achieving convergence.

Algorithm 2: DQL Algorithm for Slice Embedding (DQL-SE).

initialize: hyperparameter, environment, neural network model;
foreach *slice* $k \in \mathcal{K}$ **do**
  foreach *episodes* **do**
    Initialize initial state and action;
    while *not all states are explored* **do**
      Choose $s_t \in \mathcal{S}$;
      while *action satisfies constraints* **do**
        Randomly select action with probability ε. With probability $1-\varepsilon$: Predict Q-value for state s_t using evaluation network then choose action α_t with highest Q-value;
      Calculate reward for first state and other states: $\mathscr{R}_t = M - \left(a_{i',t} - r_{v'}\right) - \beta h_{i,i'}$;
      Save s_t, α_t, r_t, s_{t+1} to memory buffer;
      Train evaluation network, update weights, bias and Q-value: $Q(s_t, \alpha_t) \leftarrow (1-\lambda)\, Q(s_t, \alpha_t) + \lambda \left(r_t + \gamma \max_\alpha Q(s_{t+1}, \alpha_{t+1})\right)$;
    Update exploration rate: $\varepsilon = \varepsilon \times \varepsilon_{\text{decay}}$;
    if *episode % target_update_freq == 0* **then**
      Update target network weights from evaluation network;
  Find the optimal policy based on the Q-value predicted by the evaluation network;
  Update available resources on $\mathcal{G}$;

5 PERFORMANCE EVALUATION

5.1 Simulation Setup

In this section, we compare the performance of three different approaches: ILP-SE, QL-SE, and DQL-SE on solving the slice embedding problem with different settings. For each setting, we let ILP-SE run once, QL-SE and DQL-SE run five times then calculate the mean and standard deviation of the performance metrics of these two approaches. Two metrics are used for the evaluation: the acceptance rate, given by $\sum_{k \in \mathcal{K}} \frac{x_k}{|\mathcal{K}|}$, and the computing time required for solving the problem.

Physical graphs. We used two physical network topologies, the atlanta (small) and pioro40 (large) network provided by SNDlib, a widely used library for network simulation [9]. The number of

nodes and links[1] ($|\mathcal{N}|, |\mathcal{L}|$) of the atlanta and pioro40 network are respectively (15, 44) and (40, 178). The initial value of available resources at each physical node and link is set to 100.

Slices sets. For list of slices graph, we generate them randomly with four parameters: the number of slices to map , the number of virtual nodes in each slice, the required amount of resources of each virtual nodes, the required amount of resources of each virtual links. Simulations are performed with three slice settings: $|\mathcal{K}| = 100$, 200, and 300 slices, each of which has 10 nodes, and each virtual node and virtual link has a resource requirement of 2.

Q-learning parameters. Required parameters for QL-SE and DQL-SE algorithms are shown in Table 2.

Table 2: QL-SE and DQL-SE simulation parameters.

Parameter	QL-SE	DQL-SE
#episodes	1000	2100
λ	0.009	0.006
γ	0.8	0.8
ε	1.0	1.0
$\varepsilon_{\text{decay}}$	0.9954	0.9976
ε_{min}	0.01	0.01
update frequency (#episodes)	–	10
batch size	–	32
hidden layer dimension	–	32
replay buffer size	–	2000

For the DQL-SE algorithm, the neural network's architecture is shown in Fig. 2. We use a fully connected neural network with three layers. The input layer consists of $|\mathcal{N}_k|$ neurons, which is the number of VNFs of slice k. The output layer comprises $|\mathcal{N}|$ neurons, which is equal to the number of physical nodes in $\mathcal{G}$. The hidden layer between the input and output layers has 32 neurons. The weights and biases of the neural network are randomly initialized within the range $[0, 1]$ to simplify the training process. For the output layer, we use $\texttt{ReLU}(x) = \max(x, 0)$ and $\texttt{softmax}(x) = e^x / \sum e^x$ as the activation functions for the neural network output.

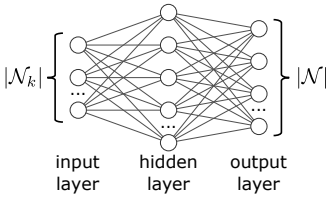

Figure 2: Detailed architecture of the neural network used in the DQL-SE.

Software, hardware and test setup. All simulations are performed with Python 3.7 on a PC with Intel i5-3320M CPU and 16 GB of RAM. The CPLEX MILP solver v12.10 is used to solve ILP-SE. QL-SE and DQL-SE are implemented from scratch by using the numpy library of Python.

[1]The atlanta and pioro40 graphs from SNDlib are originally undirected and have respectively 22 and 89 links. They are converted to directed graphs by replacing each undirected link with two directed ones with the help from NetworkX library in Python.

5.2 Results

Fig. 3 illustrates the acceptance rate of ILP-SE, QL-SE, and DQL-SE obtained from the simulations using atlanta and pioro40 network topology. Fig. 4 shows the time required to return the embedding results in the corresponding tests of each algorithm.

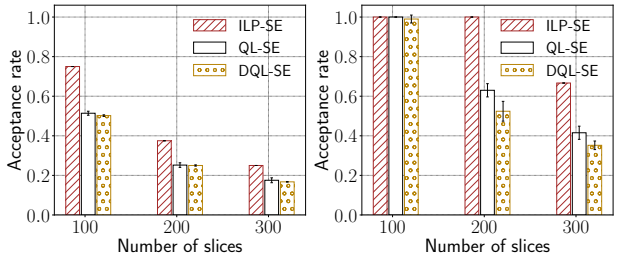

Figure 3: Acceptance rate of ILP-SE, QL-SE, and DQL-SE obtained using atlanta and pioro40 network topology.

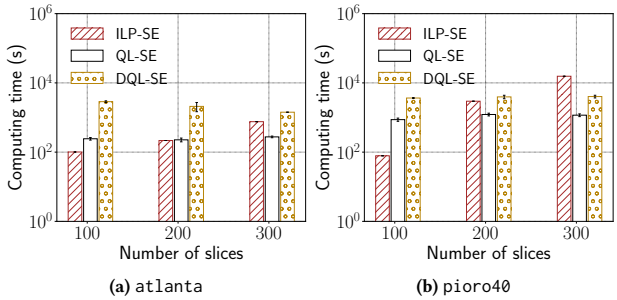

Figure 4: Computing time of ILP-SE, QL-SE, and DQL-SE obtained using atlanta and pioro40 network topology.

On one hand, the acceptance rates drop when increasing the number of slices to be embedded in both physical graphs. In general, as expected, ILP-SE outperforms QL-SE and DQL-SE in terms of acceptance rate. When dealing with unsaturated network, QL-SE and DQL-SE are able to achieve a close performance to ILP-SE, with nearly 100% of successfully embedded slices (see Fig. 3b). The performance difference becomes more significant when dealing with bigger problem instance. For instance, with pioro40 and $|\mathcal{K}| = 200$, ILP-SE is still able to achieve a acceptance rate of 100%, whereas that of QL-SE and DQL-SE drop to respectively around 63% and 52% (see Fig. 3b).

On the other hand, when dealing with larger problem size, the computing time required by ILP-SE significantly increase, due to its *NP*-hardness (see Fig. 4), whereas the increase in terms of computing time of QL-SE and DQL-SE is not significant.

Overall, QL-SE and DQL-SE are able to achieve at least half of the performance in terms of acceptance rate yielded by ILP-SE. With further development, these algorithms have the potential to become some of the best reinforcement-learning-based graph mapping algorithms in networking.

It is worth noting that, as described in Sections 4.2 and 4.3, QL-SE and DQL-SE both have a large number of matrix operations, which consumes a large amount of time if using a normal CPU. With the ability of processing matrix operations in a short amount of time [11], GPUs and TPUs can be used as a dedicated devices to accelerate the speed of both QL-SE and DQL-SE, thus make them more efficient to solve the slice embedding problem in real time.

Finally, Fig. 5 shows the evolution of Q-value of QL-SE and DQL-SE. It can be seen that the Q-value of QL-SE and DQL-SE on both `atlanta` and `pioro40` network reaches convergence after around 1000 episodes. This demonstrates the accuracy of the results yielded by QL-SE and DQL-SE.

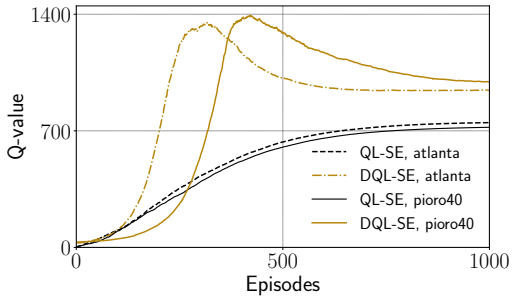

Figure 5: Evolution of Q-value with QL-SE and DQL-SE.

6 CONCLUSION

This paper considered the network slice embedding problem with the goal of maximizing the number of network slices that can be embedded into the physical network. We proposed a RL and DRL-based heuristic to address the limitations of exact mathematical approaches, which require exponential computation time as the size of the problem grows. This method is intended to overcome the disadvantages of exact mathematical methods in terms of the exponential computation time required as the size of the problem grows.

Results from extensive simulations demonstrate the efficacy of our approach. Our findings show that the proposed algorithms, QL-SE and DQL-SE have a predictable and stable computation time.

Possible improvements include the use of dedicated computing devices such as GPUs and TPUs to accelerate the speed of QL-SE and DQL-SE. With this improvement, the proposed algorithms will be able to solve efficiently the problem of network slice embedding at large-scale settings. Several extensions of this work are also possible. In future works, we shall study other DRL architectures, e.g., Deep Deterministic Policy Gradient (DDPG) [8] or Dueling double deep Q-learning [4], based on the same RL environment that has well been defined in this work.

REFERENCES

[1] NGMN Alliance. 2016. Description of network slicing concept. *NGMN 5G P*, 1, 1, 1–11.

[2] Jose Jurandir Alves Esteves, Amina Boubendir, Fabrice Guillemin, and Pierre Sens. 2022. On the robustness of controlled deep reinforcement learning for slice placement. *Journal of Network and Systems Management*, 30, 3, 43.

[3] José Jurandir Alves Esteves, Amina Boubendir, Fabrice Guillemin, and Pierre Sens. 2022. A heuristically assisted deep reinforcement learning approach for network slice placement. *IEEE Transactions on Network and Service Management*, 19, 4, 4794–4806. DOI: 10.1109/TNSM.2021.3132103.

[4] Thomas Kwantwi, Guolin Sun, Noble Arden Elorm Kuadey, Gerald Maale, and Guisong Liu. 2023. Blockchain-based computing resource trading in autonomous multi-access edge network slicing: a dueling double deep q-learning approach. *IEEE Transactions on Network and Service Management*.

[5] Xin Li, Mohammed Samaka, H Anthony Chan, Deval Bhamare, Lav Gupta, Chengcheng Guo, and Raj Jain. 2017. Network slicing for 5g: challenges and opportunities. *IEEE Internet Computing*, 21, 5, 20–27.

[6] Quang-Trung Luu, Sylvaine Kerboeuf, Alexandre Mouradian, and Michel Kieffer. 2020. A coverage-aware resource provisioning method for network slicing. *IEEE/ACM Transactions on Networking*, 28, 6, 2393–2406. DOI: 10.1109/TNET.2020.3019098.

[7] Quang-Trung Luu, Michel Kieffer, Alexandre Mouradian, and Sylvaine Kerboeuf. 2018. Aggregated resource provisioning for network slices. In *2018 IEEE Global Communications Conference (GLOBECOM)*, 1–6. DOI: 10.1109/GLOCOM.2018.8648039.

[8] Tianle Mai, Haipeng Yao, Ni Zhang, Wenji He, Dong Guo, and Mohsen Guizani. 2021. Transfer reinforcement learning aided distributed network slicing optimization in industrial iot. *IEEE Transactions on Industrial Informatics*, 18, 6, 4308–4316.

[9] Sebastian Orlowski, Roland Wessäly, Michal Pióro, and Artur Tomaszewski. 2010. Sndlib 1.0—survivable network design library. *Networks: An International Journal*, 55, 3, 276–286.

[10] Roberto Riggio, Abbas Bradai, Davit Harutyunyan, Tinku Rasheed, and Toufik Ahmed. 2016. Scheduling wireless virtual networks functions. *IEEE Transactions on Network and Service Management*, 13, 2, 240–252. DOI: 10.1109/TNSM.2016.2549563.

[11] Marc Rothmann and Mario Porrmann. 2022. A survey of domain-specific architectures for reinforcement learning. *IEEE Access*, 10, 13753–13767. DOI: 10.1109/ACCESS.2022.3146518.

[12] Saibharath S., Sudeepta Mishra, and Chittaranjan Hota. 2023. Joint qos and energy-efficient resource allocation and scheduling in 5g network slicing. *Computer Communications*, 202, 110–123. DOI: https://doi.org/10.1016/j.comcom.2023.02.009.

[13] Meryem Simsek, Oner Orhan, Marcel Nassar, Oguz Elibol, and Hosein Nikopour. 2021. Iab topology design: a graph embedding and deep reinforcement learning approach. *IEEE Communications Letters*, 25, 2, 489–493. DOI: 10.1109/LCOMM.2020.3029513.

[14] Penghao Sun, Julong Lan, Junfei Li, Zehua Guo, and Yuxiang Hu. 2021. Combining deep reinforcement learning with graph neural networks for optimal vnf placement. *IEEE Communications Letters*, 25, 1, 176–180. DOI: 10.1109/LCOMM.2020.3025298.

[15] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[16] Tianfu Wang, Qilin Fan, Xiuhua Li, Xu Zhang, Qingyu Xiong, Shu Fu, and Min Gao. 2021. Drl-sfcp: adaptive service function chains placement with deep reinforcement learning. In *ICC 2021 - IEEE International Conference on Communications*, 1–6. DOI: 10.1109/ICC42927.2021.9500964.

[17] Yikai Xiao, Qixia Zhang, Fangming Liu, Jia Wang, Miao Zhao, Zhongxing Zhang, and Jiaxing Zhang. 2019. Nfvdeep: adaptive online service function chain deployment with deep reinforcement learning. In *Proceedings of the International Symposium on Quality of Service*, 1–10.

[18] Haipeng Yao, Sihan Ma, Jingjing Wang, Peiying Zhang, Chunxiao Jiang, and Song Guo. 2020. A continuous-decision virtual network embedding scheme relying on reinforcement learning. *IEEE Transactions on Network and Service Management*, 17, 2, 864–875. DOI: 10.1109/TNSM.2020.2971543.

[19] Peiying Zhang, Chao Wang, Chunxiao Jiang, Neeraj Kumar, and Qinghua Lu. 2022. Resource management and security scheme of icpss and iot based on vne algorithm. *IEEE Internet of Things Journal*, 9, 22, 22071–22080. DOI: 10.1109/JIOT.2021.3068158.

[20] Shunliang Zhang. 2019. An overview of network slicing for 5g. *IEEE Wireless Communications*, 26, 3, 111–117.