A Novel Deep Q-Network-Based Scheme for Online Virtual Link Embedding in Software Defined Networks

Xiaodong Tan[1,2], Jinxin Du[2], Lunde Chen[2,*], Wanyu Liu[1,2]
[1]School of Mechatronic Engineering and Automation of Shanghai University, Shanghai, 200444 China
[2]Sino-European School of Technology of Shanghai University (UTSEUS), Shanghai, 200444 China
*Corresponding author: Lunde Chen (email: lundechen@shu.edu.cn)

***Abstract*—With the advent of Software Defined Networking (SDN) and network virtualization, network systems are becoming increasingly more flexible and more scalable. With a global view and a centralized control, SDN can assign paths for individual virtual link (VL) requests, by employing classic graph algorithms (e.g. Dijkstra's algorithm). However, this presumes that VL requests are independent of each other, which doesn't hold true in real-world scenarios where VL requests can arrive randomly. In fact, previously instantiated VLs can take up resources (e.g. bandwidth) in the substrate network and render the residual resources unbalanced, leading to sub-optimal path assignment for subsequent VLs. This work investigates such scenarios and proposes a novel scheme based on Deep Q Network (DQN) and Link Feature Representation (LFR) for efficient online VL embedding. Experiments show that our proposed scheme (DQN-LFR) can not only efficiently and effectively meet the routing requests of various VLs, but also ensure the balance of virtual network resources, leading to a higher acceptance rate. Moreover, the proposed solution has also the advantage of self-adaptability, scalability and generalization ability.**

***Index Terms*—Software Defined Networking, Deep Q Network, Link Feature Representation, Network Embedding, Load Balancing**

I. Introduction

With increasing exigence from emerging network services, new Internet architecture and control strategies have been proposed for an improved Quality of Service (QoS) and Quality of experience (QoE), such as SDN [?] and Network Functions Virtualization (NFV) [?]. SDN has revolutionized the whole network landscape, enjoying huge success in data centers and telecommunication operators [?]. By separating control plane from data plane, SDN centralizes control on the controller, and frees the constraints of the underlying hardware on the network architecture. With SDN, network virtualization has become straightforward, as the physical network can be abstracted into a network model [?], through which a global view of network topology is established. The controller can take advantage of this global view to make decisions and allocate shared resources in the entire network according to user needs [?].

With dynamic user requests and constantly changing substrate physical network conditions, it becomes increasingly difficult to develop a decent VL embedding scheme [?]. Existing methods can barely achieve both solution accuracy and computation efficiency at the same time. As a matter of fact, both the methodology and feature representation of the network have significant impact on the efficiency and effectiveness. Without being exhaustive, some prominent related works are summarized in Table I. As we can see, feature representation of network models can be classified mainly in two categories: Link Feature Representation (LFR) and Path Feature Representation (PFR). LFR represents the network model with the underlying network's links and nodes, and ensures the solution accuracy by strictly respecting the resource constraints, but its computation efficiency is difficult to guarantee. Integer programming coupled with LFR was employed in [?] and [?] to obtain accurate solutions, but their computation time can be high when the network size is large. Therefore, it suffers from poor scalability. On the other hand, PFR is constructed from the path level, and provides a matrix containing "k" candidate paths, hence won't need to consider the constraints of each node and link (the controller only needs to select one among the "k" candidate paths), which could largely reduce the computation time, but at the cost of reduced accuracy. This latter representation has been generally used in heuristics, such as [?] and [?].

With the advent of AI, researchers have been using machine learning techniques for virtual network embedding. Yuan et al. [?] used Q-learning algorithm to obtain an optimal control strategy, which performed well in simple scenarios; Stampa et al. [?] used Deep Deterministic Policy Gradient (DDPG) coupled with PFR, in which the agent can make its own decision and choose the best one from the candidate paths. However, due to the intrinsic limitation of PFR, the proposed methods cannot learn the structural features of the network, as a result, it has sub-optimal solution quality and shows poor generalization ability. In addition, it is necessary to compute in advance the matrix of candidate paths of each source-destination pair, making it inadequate in dynamically changing network environments. Unfortunately, most other Deep Reinforcement Learning (DRL) based methods [?] [?] employed PFR, and suffered from the same limitations.

This work tackles the SDN embedding problem with DQN, which provides the fitting ability of neural networks and the decision-making ability of Reinforcement Learning (RL). However, unlike other DQN-based solutions in the literature,

TABLE I

A NON-EXHAUSTIVE CLASSIFICATION OF SDN VIRTUAL NETWORK EMBEDDING SCHEMES

Author	Methodology	Feature Representation	Quality of solution	Computation efficiency
Stasi et al. [?]	ILP	LFR	Optimal	Affordable
Dehury et al. [?]	ILP, GA	LFR	Optimal for ILP, Suboptimal for GA	Affordable
Yun et al. [?]	Heuristic	PFR	Suboptimal	Acceptable
Wang et al. [?]	Greedy	PFR	Suboptimal	Acceptable
Yuan et al. [?]	Q-learning	PFR	Suboptimal	Costly due to huge solution complexity
Stampa et al. [?]	DDPG	PFR	Suboptimal	Acceptable
Suare et al. [?]	Deep RL	PFR	Suboptimal	Acceptable
Chen et al. [?]	Deep-RMSA	PFR	Suboptimal	Acceptable

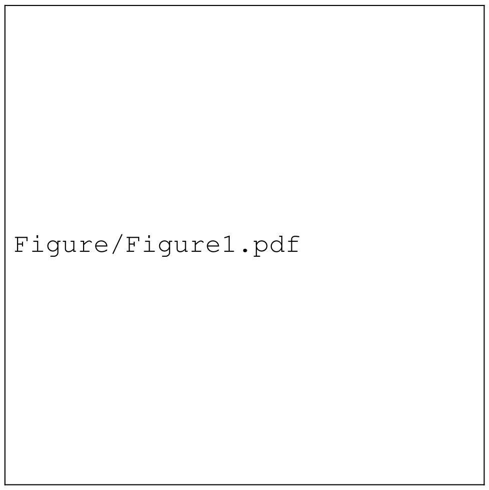

Fig. 1. An illustration of network and link request model

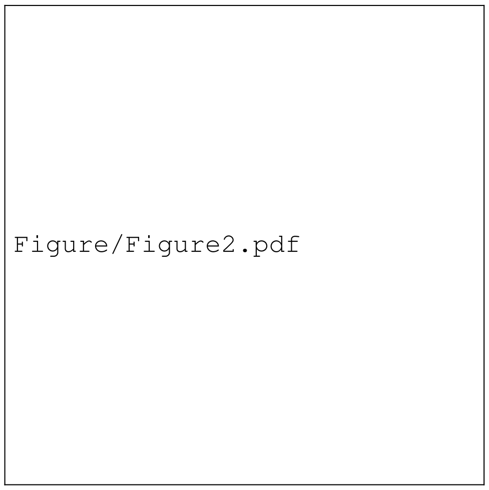

Fig. 2. An illustration of load balancing

this work uses LFR for the network model presentation, which can more accurately represent the structural characteristics of the network from the underlying resources such as nodes, link cost and bandwidth. To the best of our knowledge, this is the first work of network embedding in which LFR is used in tandem with DQN. By incorporating LFR, DQN agents can learn the underlying network structure to ensure a high solution quality, with a very low computation time at inference phase. Moreover, in our proposed scheme, the end-to-end nature of DQN can enhance the generalization and adaptive capabilities of the model. After training, the agent can instantly make decisions in complex and unknown scenarios and continuously optimize to achieve balanced allocation of network resources and meet the network service quality requirements of users.

This paper is organized as follows. Section II introduces the system model used in our formulations. Section III describes our proposed DQN-LFR for VL embedding. Section IV presents the performance analysis of the proposed method. Finally, Section V concludes the paper.

II. SDN VIRTUAL NETWORK MODEL

A. Network Model

In a SDN virtual network architecture, the requests initiated by users at the application layer are instantiated by the controller at the infrastructure layer. The physical model of the SDN network is denoted by $G = (V, E)$, where G is a directed graph representing the SDN network, V is the set of nodes and E is the set of half-duplex bidirectional links. The set of nodes is represented as $V = (v_1, v_2, ..., v_i) = ((x_1, y_1), (x_2, y_2), ..., (x_i, y_i))$, where x_i, y_i are the location of the node V_i. The set of nodes is denoted by $E = (e_1, e_2, ..., e_j) = ((w_1, b_1), (w_2, b_2), ..., (w_j, b_j))$, where w_j is the transmission cost and b_j is the residual bandwidth resource of link E_j. For a node v, the set of its adjacent neighbour nodes is noted as N_v.

The link request model is denoted by $k = ((\nu, \mu), b_k)$, where ν is the source node, μ is the destination node and b_k is the bandwidth consumed by the request k. Fig. 1 shows an example of a user request $k = ((A, D), 1)$, i.e. from node A to node D with a bandwidth of 1. If the instantiated VL takes the path $A \rightarrow B \rightarrow D$, then afterwards link (A, B) and link (B, D) will have a residual bandwidth of 6 and 16, respectively.

B. Load Balancing Model

When processing virtual link requests, SDN controllers typically use shortest path algorithms such as Dijkstra's algorithm or its extended versions [?] to find an optimal solution. This applies to user requests that are initiated and terminated se-

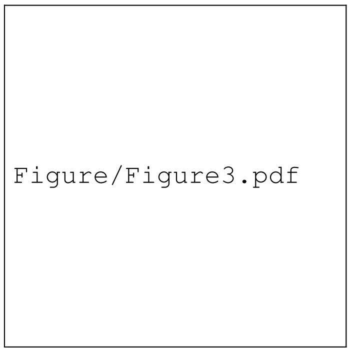

Fig. 3. An illustration of action state space for the network model

quentially, i.e. only after one request ends will the next request arrive. However, in practice, user requests are usually random and difficult to predict. Deriving the optimal solution only from the current state cannot guarantee the global optimality.

Fig. 2 illustrates two ways to embed a request $k_1 = ((A, C), 1)$: the link cost of $path1 = (A \rightarrow B \rightarrow C)$ (in black) is $w_1 = 3$, and the link cost of $path2 = (A \rightarrow E \rightarrow C)$ (in orange) is $w_2 = 4$. Although the cost of $path1$ is lower, it causes the bandwidth of link (B, C) to be exhausted and unable to admit new requests. If a new request $k_2 = ((D, C), 1)$ arrives, no eligible path for embedding k_2 will be available. This is due to the fact that the load is not balanced. By contrast, if $path2$ is chosen as the embedding path, it would have better load balancing and can further admit new requests, e.g. k_2. Therefore, to augment the acceptance rate of VLs, it's desirable to have a balanced residual bandwidth.

III. DQN-LFR Algorithm for VL Embedding

DQN can solve decision-making problems in complex scenarios and has enjoyed huge successes with Alpha Go [**?**] and other games such as Atari [**?**]. In DQN, agents can learn optimal strategies to maximize cumulative rewards by iteratively exploring states and action spaces [19]. The learning process of the DQN algorithm can be abstracted into Markov quadruples (S, A, R, S') [**?**], representing states, actions, rewards, and next states, respectively. This work uses the improved DQN [**?**], which incorporates experience pool playback and asynchronous update of the network to break the correlation between training input. It is further enhanced by incorporating LFR so that the DQN agent can learn underlying SDN network features. As a matter of fact, feature representation is essential to the DQN agent. A good feature representation can enable the algorithm to learn the structural features of the network model, leading to better convergence and improved solution quality. Hereafter we detail our proposed DQN-LFR:

1) State space S: The DQN agent obtains state from the environment. The state information will then serve as the input of the neural network feature extraction stage [**?**]. In our formulation, we represent the state of the DQN agent with the position of the node. As shown in Fig. 3, if the DQN agent is located at the node (or, state) s_1, and the state transition process can be either (a_1, s_2) or (a_2, s_3). In other words, with DQN agent currently at s_1, if action a_1 is performed, then the DQN agent will transit to s_2; if action a_2 is performed, then the DQN agent will transit to s_3. We see that there is a bi-injection between the node position and the state. Therefore, the state space of the DQN agent can be represented as shown in Equation 1:

$$S = \left\{s_1, s_2, ..., s_{|V|}\right\} \tag{1}$$

2) Action space A: The action space of the DQN agent corresponds to the set of states of its adjacent nodes N_{curr}. More specifically, each action $a = (s_{curr}, s_{next})$ represents the transition to be carried out from the current node (state) s_{curr} to another state $s_{next} \in N_{curr}$. As shown in Fig. 3, with the DQN agent in state s_1, only action a_1 or action a_2 can be executed. The action space of the agent can hence be defined as shown in Equation 2:

$$\begin{aligned} A &= \{a_1, a_2, ..., a_{|N_{curr}|}\} \\ &= \{(s_{curr}, s_{n_1}), (s_{curr}, s_{n_2}), ..., (s_{curr}, s_{n_{|N_{curr}|}})\} \end{aligned} \tag{2}$$

3) Reward function R: The reward of the DQN agent indicates the quality of the action. A leading-to-success action will return a positive reward, while a leading-to-failure action will return a negative reward. At each state, the DQN agent can perform different actions, leading itself to different subsequent states, accumulating rewards at each transition. To differentiate between "slightly good" and "really good" actions, the reward function needs to be designed appropriately. In this work, the reward function are composed of two parts: single-step reward and endpoint reward. The single-step reward is what the DQN agent gets as a reward after executing one action. This reward is introduced to mitigate the problem of reward sparsity of reinforcement learning. The single-step reward is defined as shown in Equation 3:

$$r_i = \alpha w_i + \beta b_i \tag{3}$$

where α and β are the weight coefficient of the linear combination of link cost and residual bandwidth associated to the link $e = (v_{curr}, v_{next})$, or, bi-injectively, $a = (s_{curr}, s_{next})$. The second part of the reward function, i.e. endpoint reward, can guide the DQN agent to reach the destination node. Apparently, this latter reward should be much more significant than the single-step reward, because ultimately, the goal of a path is to reach the destination node. The endpoint reward c is defined as a constant, as shown in Equation 4:

$$r_n = c \tag{4}$$

The reward function R for the agent can therefore be defined as shown in Equation 5:

$$\begin{aligned} R &= \sum_{i}^{n-1} r_i + r_n \\ &= \sum_{i}^{n-1} (\alpha w_i + \beta b_i) + c \end{aligned} \tag{5}$$

We can see that in our DQN-LFR, for representing state space S, action space A and reward space R, the network model (nodes, edges, residual bandwidth, costs, etc.) is at the central position and is used all the time. Powered with LFR, the DQN agent can continuously learn the structural features and resource features of the substrate network, which is key to our proposed DQN-LFR. With such rich information of the network, we can naturally expect DQN-LFR to perform better for embedding VLs, compared to other techniques that don't take such detailed information into formulation.

IV. Performance evaluations for embedding

The objective of this performance analysis is to show that our proposed DQN-LFR scheme for VL embedding clearly has the advantage of being computationally efficient, while being able to effectively deliver near-optimal solutions. We will also discuss about its excellent adaptability and generalization ability.

A. Algorithms for Comparison

We compare the DQN-LFR algorithm with the following algorithms:

- ILP: Integer Linear Programming model adopted from [?], which takes load balancing into consideration. In the formulation, paths are constructed by conservation of flow, and load balancing is guaranteed by including a penalty term in the objective function which represents the gap between the maximum and minimum residual bandwidth of the links after embedding.
- KSP: A heuristic derived from k-shortest path algorithm [?], whose main idea is to choose, among the "k" shortest paths, the one that will give the best load balancing. We can see this algorithm incorporates a simple strategy of load balancing, while retaining the main strengths of k-short path algorithm. The pseudo code for KSP is shown in Algorithm 1.

Algorithm 1: KSP heuristic algorithm

```
Input: $G = (V, E)$, $k_{request}$
Output: $\hat{p}$
1:  $G' = (V', E') \overset{\alpha w + \beta b}{\longleftarrow} G = (V, E)$
2:  $P \longleftarrow$ kShortestPath($G', k_{request}$)
3:  $\hat{p} \longleftarrow null$
4:  $\hat{l} \longleftarrow \infty$
5:  for $p \in$ P do
6:      $l \longleftarrow$ getLoadBalancingValue($G', p$)
7:      if $l < \hat{l}$ then
8:          $\hat{l} \longleftarrow l$
9:          $\hat{p} \longleftarrow p$
10:     end if
11: end for
```

Figure/Figure4(a).pdf

(a) Cumulative reward

Figure/Figure4(b).pdf

(b) Single-step reward

Fig. 4. Training process of DQN-LFR agent

B. DQN-LFR Model

Our DQN-LFR algorithm consists of two parts: Deep Learning (DL) and RL. The DL part consists of a three-layer fully connected neural network for feature extraction and function fitting, and the RL part adopts a Q-learning algorithm based on Temporal Difference (TD). During the training phase, DQN-LFR model randomly sample data from the experience pool in batches, then the sampled data is used as the input of the network layer, whose output is used as the input of the Q-learning algorithm, and an action with the greatest Q value is chosen as the final output.

TABLE II
PARAMETERS OF NETWORK MODEL AND DQN AGENT

Parameter	Value
Network size	5*5
Number of link	40
Cost of link	$(0, 20) \cap \mathbf{N}^*$
Capacity of bandwidth	$(0, 20) \cap \mathbf{N}^*$
learning rate α	0.001
Probability of exploration	$e^{\frac{2}{n}}$
Discount factor γ	0.98
Capacity of experience pool	$\frac{n}{2}$

C. Simulation Settings and Implementation

The network model and parameter settings of the DQN-LFR used in the experiment are depicted in Table II.

1) Network model: For space reasons, one single network instance is considered in the presented results, which consisits of a directed graph without self-loops composed of 25 nodes and 40 half-duplex links, and all nodes are reachable under the condition of sufficient network resources. Each link is associated with a randomly generated transmission cost w in integer, as well as a randomly generated residual link bandwidth b in integer, with $w \in (0, 20) \cap \mathbf{N}^*$ and expressed in unity of cost (UC), $b \in (0, 20) \cap \mathbf{N}^*$ and expressed in unity of bandwidth (UB).

2) Load model: Each VL request consists of a random source and a random destination, taking up a bandwidth of 1 UB. After a VL's successful embedding, bandwidth resources are consumed, before the arrival of a next random VL request.

3) DQN-LFR parameters: For the DQN part, the parameter n refers to the number of training episodes. The experience pool capacity is set to half of the number of training episodes n. The exploration probability $\epsilon \in (0, 1)$ decays exponentially with the number of training episodes n, i.e. $\epsilon = e^{\frac{1}{\frac{n}{2}}} = e^{\frac{2}{n}}$, which will decay exponentially with the increase of training episodes, ensuring that at the beginning of training, the agent will explore the state space for the most part and won't fall into local optimum, and stops exploring after convergence or finding the optimal strategy. We set the discount factor as $\gamma = 0.98$, which indicates the importance of future rewards. Finally, the learning rate of DQN is set as $\alpha = 0.001$.

4) ILP parameters: For ILP, the penalty term for load balancing in the objective function was set to 10.

5) KSP parameters: For KSP, we used 3-shortest paths, i.e. k is set to 3.

6) Implementation: The ILP model was implemented in Python with CPLEX 12.9 solver. KSP was implemented with Python. DQN-LFR was implemented with PyTorch. NetworkX [**?**] was used to represent graphs. The experiments were carried out on an Ubuntu 20.04 server with two GPUs of Nvidia GeForce RTX 3090 Ti. Our code used for the experiment is open-sourced and hosted on GitHub [1].

[1]https://github.com/evidentiallab/dqn-lfr

D. Performance Metrics

The following metrics are used for performance analysis purposes:

1) Convergence: This metric concerns only DQN-LFR during its training phase among the three above-mentioned methods. A reinforcement learning model won't be considered as working if it doesn't converge. It's therefore important to guarantee this aspect of DQN-LFR when we train the model.

2) Empirical accuracy: This metric refers to the rate of identical solutions of DQN-LFR or KSP compared to ILP. In fact, if we assume that ILP, with decent parameters, will always give an optimal solution in its own sense, then we could take its solution result as a benchmark. The more a method gives identical solution as ILP, the more we could consider it as well-performing, in an empirical sense.

3) Average path cost: This metric refers to the average cost of computed paths for embedding. Certainly, a shortest-path algorithm will always give the best result for this metric, but could lead to unbalanced resource allocation. As all three methods take load balancing into consideration, under the assumption that a certain level of load balancing is achieved, this metric can be indicative of the solution quality.

4) Computation time: The computation time is critical for any resource embedding algorithm, as this is directly related to "time to delivery" of VL requests. This metric refers to the time needed for processing a VL request before outputting an embedding result. Note that for DQN-LFR, the computation time consists in the inference (prediction) time, not the training time.

5) Load balancing: This metric measures whether and to which extent an algorithm can give results that lead to a decent load balancing. Note that an absolutely balanced solution is not necessary, as this will often lead to higher path costs. On the other hand, if an algorithm has no load balancing functionality, earlier resource exhaustion will occur, leading to reduced acceptance rate and render the network overloaded but under-utilized.

6) Scalability: This metric refers to how an algorithm would perform when the substrate network grows in size. If an algorithm performs well (in terms of solution quality and computation efficiency) with small-sized networks but poorly with large-scale networks, then the algorithm is considered as unscalable.

7) Self-adaptability: Self-adaptability refers to an algorithm's ability to modify its own structure and behaviour in response to changes in the operating environment. This is important in a dynamic network setting where actions should be taken accordingly to maintain the quality requirements.

8) Generalization ability: This metric refers to an algorithm's ability to generalize to unseen testing environments. Ideally, a model trained with a certain environment should be also usable in another environment. Broadly speaking, generalizability could be enhanced by preventing overfitting.

TABLE III
COMPARISON RESULTS OF DQN-LFR, ILP AND KSP FOR DIFFERENT METRICS

Method	Convergence	Empirical accuracy	Average path cost	Computation efficiency	Load balancing	Scalability	Self-adaptability	Generalization ability
DQN-LFR	Yes	High	Low	High	Good	Yes	Yes	Yes
ILP	Not applicable	High	Low	Low	Good	No	No	No
KSP	Not applicable	Low	High	High	Mediocre	Yes	No	No

E. Performance Evaluation Results

1) Convergence: Fig. 4 (a) shows the cumulative reward of the agent with the increase of episodes for our proposed DQN-LFR model during the training phase. When n reaches around 10k, our model begins to converge and the cumulative reward begins to increase (e.g., in a full episode, the reward is positive when the endpoint is greater than the sum of the single-step rewards). After 10k episodes, DQN-LFR keeps on exploring and improving the strategy, before finally converging to a stable strategy.

Fig. 4 (b) shows the single-step reward (excluding the endpoint reward) of the model during the training phase, and point A shows that the DQN-LFR model is driven by exploration probabilities to explore new states to avoid falling into local optimum.

2) Empirical accuracy: Fig. 5 (a) shows the empirical accuracy of KSP and DQN-LFR. We can see that in around 82% cases, DQN-LFR gives identical results as ILP, significant higher than the 58% for KSP. This indicates that DQN-LFR gives very high quality solutions and is near-optimal.

3) Average path cost: Fig. 5 (b) shows the average path cost results. Although ILP has the best performance for this metric, DQN-LFR is just slightly behind. On the contrary, KSP has much worse performance in this aspect.

4) Computation efficiency: Fig. 5 (c) shows the total calculation time for solving a randomly generated 1000 requests: ILP takes the most time due to accurate calculations, and both the trained DQN-LFR model and KSP are completed within an acceptable time.

5) Load balancing: Although all three methods take load balancing into formulation, we observe that ILP and DQN-LFR gives better load balancing than KSP. This is expected because the strategy of load balancing incorporated into KSP is very simple, and cannot cope with complex scenarios.

Thanks to their excellent load balancing performances, ILP and DQN-LFR don't suffer from overloaded links that have no bandwidth left for admission of subsequent VLs. This partially contributes to the good solution quality of ILP and DQN-LFR with a high acceptance rate. On the contrary, with a limited load balancing capability, KSP has worse solution quality and has lower acceptance rate of VL requests.

6) Scalability: Ideally, a network embedding algorithm should be scalable in order to cope with constant-growing network communications, especially in IoT or cloud computing scenarios. However, for ILP, which already has a poor computation efficiency for small-sized networks, its computational complexity explodes when the substrate network grows in size, making it ineligible for large-sized networks.

On the contrary, DQN-LFR and KSP have excellent scalability. KSP scales well as it is just an improved version of classic k-shortest path algorithm. In the case of DQN-LFR, this is due to the fact that for calculating a VL solution, it's one simple forward propagation of the neural network, and the size of substrate network has negligible impact on the computation time.

7) Self-adaptability: With the dynamicity of substrate network and VL requests, our proposed DQN-LFR was able to self-adapt, thanks to its learning abilities in the neural network part. ILP and KSP, on the other hand, are substantially different in this regard. Due to their lack of learning ability, they can not effectively cope with changing environment in a non-stop manner and have to relaunch the calculation each time changes occur.

8) Generalization ability: Our experiments show that DQN-LFR has excellent generalization ability. In the training phase, as network models were random generated, there was a significant diversity of input data. With the learning ability of DQN-LFR, features of the network and patterns of VL requests are encoded in the neural network during its training phase, further enhancing its generalizibility. On the contrary, as ILP and KSP have no learning ability, they clearly have no generalization ability.

F. Final Verdict

The above analyses are summarized in Table III. We can see clearly that DQN-ILP performs well on all above-listed metrics. By contrast, ILP suffers from low computation efficiency, poor scalability, lack of self-adaptability and lack of generalization ability. KSP has low empirical accuracy, high average path cost and is not self-adaptable or generalizable. It is hence safe to claim that DQN-ILP has the best overall performance and gives good evaluation results on all metrics.

V. CONCLUSION

Traditional SDN virtual network embedding algorithms have high computational complexity and don't take the resource conditions of future VL requests into consideration, making them inapplicable in dynamically changing network environments. This work proposes a novel VL embedding scheme based on DQN, and by using link-level feature representation method (LFR), the scheme can efficiently meet the requirements of VL requests. Moreover, our scheme is

Figure/Figure5(a).pdf

(a) Accuracy of KSP v.s. DQN-LFR

Figure/Figure5(b).pdf

(b) Average path cost of DQN-LFR v.s. ILP and KSP

Figure/Figure5(c).pdf

(c) Computation time of DQN-LFR v.s. ILP and KSP

scalable, self-adaptable and generalizable, making it an eligible and promising AI-based resource embedding solution for telecommunication operators and cloud providers. As future work, we will extend our scheme for solving embedding problems under more constraints (latency, packet loss rate, switching resources, CPU resources, etc.), and in more complex network scenarios, e.g. wireless multi-hop multi-radio multi-channel networks.

ACKNOWLEDGEMENT

This work was supported by the Natural Science Foundation of China (NSFC) under Grant 61901253.