


ORIGINAL RESEARCH

A generic intelligent routing method using deep reinforcement learning with graph neural networks

Wanwei Huang¹ | Bo Yuan^{1,2}  | Sunan Wang³ | Jianwei Zhang¹ | Junfei Li⁴ | Xiaohui Zhang⁵

¹College of Software Engineering, Zhengzhou University of Light Industry, Zhengzhou, People's Republic of China

²The Third Construction Co. Ltd of China CREC Railway Electrification Engineering Group, Zhengzhou, People's Republic of China

³Electronic and Communication Engineering, Shen Zhen Polytechnic, Shenzhen, People's Republic of China

⁴National Digital Switching System Engineering & Technological R&D Center, Zhengzhou, People's Republic of China

⁵Henan Xinda Wangyu Technology Co. Ltd, Zhengzhou, People's Republic of China

Correspondence

Sunan Wang, Electronic and Communication Engineering, Shen Zhen Polytechnic, Shenzhen 518005, People's Republic of China.
Email: wangsunan@szpt.edu.cn

Funding information

National Natural Science Foundation of China, Grant/Award Numbers: 62002382, 62072416; The Project of Science and Technology in Henan Province, Grant/Award Numbers: 222102210175, 222102210111; Postgraduate Education Reform and Quality Improvement Project of Henan Province, Grant/Award Number: YJS2022AL035

Abstract

Routing optimization is a well-known and established topic with the fundamental goal of operating networks efficiently. Traditional optimization heuristics may suffer from performance penalty as it mismatches actual traffic, while artificial intelligence (AI) which has undergone a renaissance recently is gradually being applied to the network optimization and has shown excellent advantages. Especially deep reinforcement learning (DRL) is investigated as a key technology for routing optimization with the goal of enabling networks self-driving. Therefore, we contributed in this paper a novel approach for practical intelligent routing method using DRL with GNN, which could be easily implemented as a northbound application on the SDN controller. Our method can not only output continuous control actions for routing optimization but also learn from some networks and generalize to other unseen ones. In order to emphasize the generalization and practicality of the intelligent routing method, we deploy it in a real SDN network for experimentation rather than simulation. The results show that the method can keep on optimizing the routing of traffic in other networks of different topologies after the training is stable. And compared with hop-based OSPF, the optimal load-balancing algorithm and the recent intelligent routing DROM, it reduces network delay by 16.1%, 19.6% and 14.3%, respectively, but at the expense of flow-table space within the acceptable range.

1 | INTRODUCTION

Traffic optimization, for example, flow scheduling [1, 2], load balancing [3, 4], and routing [5], is a well-known and established topic with the fundamental goal of operating networks efficiently, in which routing is the most critical influencing factor. As Google's research [6], links in a private WAN connecting the datacenters can achieve near 100% utilization by splitting application flows among multiple paths to balance capacity against application priority/demands. Traditionally, routing optimization is dependent on handcrafted heuristics for varying traffic

scenarios, but the heuristics method is difficult to adapt to large-scale networks and may suffer performance penalty as it mismatches actual traffic [2, 7]. As Artificial Intelligence (AI) has undergone a renaissance recently and made major progress in many key domains [8, 9], it is also gradually being applied to the modelling and optimization of the network and has shown excellent results. For example, [10] presents Deep-Q to learn the QoS model directly from traffic data using Deep Generative Network, which achieves on average 3× higher inference accuracy than traditional queuing-theory-based solution; [11] designs DeepConf to automate datacentre network

This is an open access article under the terms of the [Creative Commons Attribution-NonCommercial-NoDerivs](https://creativecommons.org/licenses/by-nc-nd/4.0/) License, which permits use and distribution in any medium, provided the original work is properly cited, the use is non-commercial and no modifications or adaptations are made.

© 2022 The Authors. *IET Communications* published by John Wiley & Sons Ltd on behalf of The Institution of Engineering and Technology.

topologies management with machine learning, which performs comparably to the optimal solution.

Knowledge-defined network (KDN) is the current research hotspot [12], that is, the application of artificial intelligence technology to the network. Especially deep reinforcement learning (DRL), which is widely used in decision-making and automated control problem [13], is investigated as a key technology for routing optimization with the goal of enabling networks self-driving. [14] proposes a DDPG (deep deterministic policy gradient, a DRL algorithm which is suitable for solving optimization problems of continuous control [15]) agent that automatically adjusts links' weight related to the shortest path to minimize the network delay. But the DRL-based solution fails to generalize when applied to different network scenarios, that is, facing a network state not seen during training. In addition, [16] proposes a DQN + GNN (graph neural networks, a type of neural networks which can support relational reasoning and combinatorial generalization [17]) architecture that is able to optimize and generalize over arbitrary network topologies on an SDN-based optical transport network (OTN) scenario. However, since the action space of this method is discrete and limited, zero padding is required to unify the output dimension of the neural network, which results it only effective in simple networks such as OTN. Furthermore, [18] proposes a generalizable data-driven routing method with continuous control to allocate flows on multiple-paths arbitrarily, but its routing strategy is too complicated and requires a large flow-table space [19]. In summary, the existing studies still stay in theory, which are difficult to be applied in real networks as lack of considering the constraints such as traffic demands, flow-table size, or consistency update, and so forth.

Therefore, we are aimed to explore a practical intelligent routing method using DRL with GNN, which could be easily implemented as a northbound application on the SDN controller to optimize traffic. For this purpose, we need to consider the advantages of AI technology and the constraints of the network environment, and innovate in practice. In this paper, we make the following contributions:

- Input: We use the state that is easy to collect from the switch as the input of DRL, rather than the traffic demand that needs to be inquired from the terminal in advance. The state refers to the statistics of port, queue, flow-table, link, and so forth, which could be collected through OpenFlow or Inband Network Telemetry (INT). But the traffic demand is usually inaccurate and difficult to determine in the Internet.

- Process: We use proximal policy optimization (PPO), a DRL algorithm which outperforms other online policy gradient methods [20]) combined with GNN as the intelligent agent to process the above complex state. The PPO + GNN agent can not only output continuous control actions but also learn from some network topologies and generalize to other unseen ones, which are two important characteristics for routing optimization.

- Output: We perform post-processing on the actions of the agent, facilitating the output of an operable flow-table. The post-processing may compromise some optimizations, but

makes the flow-table easier to operate, smaller to store and more reasonable to update in the switch.

The rest of this paper is organized as follows. Section 2 reviews the background of PPO and GNN. Section 3 specifies the routing optimization problem. Section 4 presents the design of the GNN-based DRL agent. Section 5 introduces experimental environment and gives the experiment results. Finally, Section 6 concludes this paper and suggests extensions to this work.

2 | BACKGROUND

2.1 | Proximal policy optimization

PPO is a random policy-based DRL algorithm proposed by OpenAI in 2017[20]. It has good performance, especially for continuous control problems, so it is the best implementation of the current policy gradient algorithm. The derivation of the PPO algorithm requires knowledge of policy gradient [21], Actor-Critic architecture [22] and TRPO algorithm [23], which is boring and difficult to understand for scholars studying networks. To simplify, we will only describe the key principles of the Actor-Critic architecture on which the PPO algorithm relies, which is sufficient for us to apply it.

In DRL algorithm, total discount future reward is defined as:

$$R = r_0 + \gamma r_1 + \gamma^2 r_2 + \dots + \gamma^n r_n \quad (1)$$

where γ represents discount factor and r represents the reward in an interaction. An intuitive policy objective function will be the expectation of the total discount reward, expressed as:

$$L(\theta) = E[r_0 + \gamma r_1 + \gamma^2 r_2 + \dots | \pi_\theta(s, a)] \quad (2)$$

where $\pi_\theta(s, a)$ represents an actor neural network with parameter θ , which is the probability of outputting the action a in the state s . Then we can transform (2) with (1) into

$$L(\theta) = E_{s \sim p(s|\theta)}[R] \quad (3)$$

Now in the continuous case, we can use the SARSA formula $Q(s_t, a_t) = R_{t+1}$. Furthermore, we can write the gradient of a deterministic policy $a = \mu(s)$ as:

$$\frac{\partial L(\theta)}{\partial \theta} = E_{s \sim p(s|\theta)} \left[\frac{\partial Q}{\partial \theta} \right] \quad (4)$$

Applying the chain rule:

$$\begin{aligned} \frac{\partial L(\theta)}{\partial \theta} &= E_{s \sim p(s|\theta)} \left[\frac{\partial Q^\theta(s, a)}{\partial a} \frac{\partial a}{\partial \theta} \right] \Rightarrow \frac{\partial L(\theta)}{\partial \theta} \\ &= \frac{\partial Q^\theta(s, a, \omega)}{\partial a} \frac{\partial a}{\partial \theta} \end{aligned} \quad (5)$$

where the policy parameters θ can be updated via stochastic gradient ascent. Therefore, in the Actor-Critic architecture, we can train the *Actor* according to (5) with the partial derivative in *Critic*, and train the *Critic* according to the basic loss function:

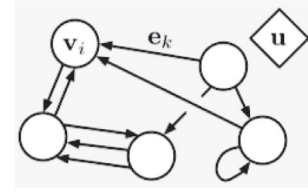
$$loss = [r + \gamma Q(s', a') - Q(s, a)]^2 \quad (6)$$

2.2 | Graph neural networks

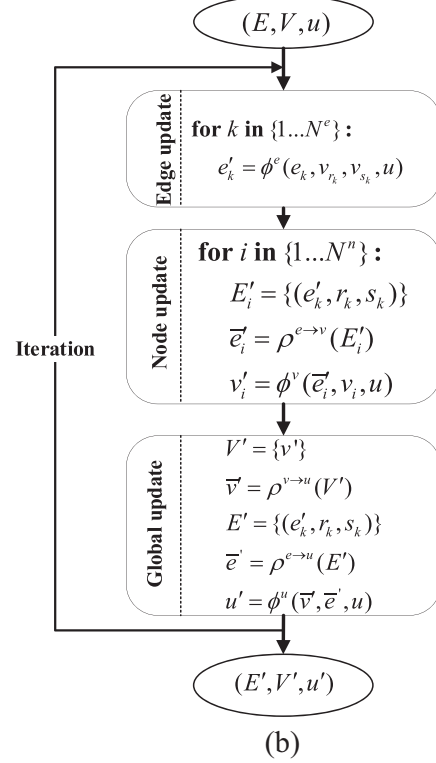
Neural networks that operate on graphs, and structure their computations accordingly, have been developed and explored extensively for more than a decade under the umbrella of ‘graph neural networks’ but have grown rapidly in scope and popularity in recent years. There are many models in the GNN family, such as message-passing neural network (MPNN) [24], non-local neural networks (NLNN) [25], and so forth. The most popular of them is *graph networks* (GN) framework proposed by Battaglia et al. [17, 26], which can be translated from other GNN models. The main unit of GN framework is the GN *block* shown in Figure 1a, which is a directed, multi-graph with global attribute, defined as a 3-tuple $G = (E, V, u)$, where E is the set of attributes for all edges, V is the set of attributes for all nodes and u is a global attribute. The steps for the GN block to update these attributes are shown in Figure 1b.

When a graph G is provided as the input to a GN *block*, the computations proceed from the edge to the node, and then to the global level. Finally, it outputs another graph G' with different attributes. These attributes are the features of nodes and edges, expressed in tensors of fixed dimensions. A GN block contains three ‘update’ functions ϕ (ϕ^e, ϕ^v, ϕ^u) and three ‘aggregation’ functions ρ ($\rho^{e \rightarrow v}, \rho^{v \rightarrow u}, \rho^{e \rightarrow u}$), where each ϕ is usually a function expressed by a deep neural network and each ρ must be the functions that are invariant to permutations of their inputs, and should take variable numbers of arguments (e.g. *Max*, *Sum*, *mean*, etc.). GN can loop through the above process until the inference error of a class of attributes (not necessarily all attributes) that we care about reaches a certain level [17].

As GNs do not only perform computations strictly at the global level (represented as u), but also apply shared computations across the node entities (represented as v_i) and across the node relations (represented as e_k) as well, their structure naturally supports combinatorial generalization. This allows never-before-seen systems to be reasoned about, because they are built from familiar components, in a way that reflects von Humboldt’s ‘infinite use of finite means’. And, a lot of studies have verified its capacity for combinatorial generalization [26]. Therefore, many traditional computer science problems, which involve reasoning about discrete entities and structure, have also been explored with graph neural networks, such as combinatorial optimization [27], Boolean satisfiability [28], and performing inference in graphical models [29].



(a)



(b)

FIGURE 1 GN block. (a) The definition of GN, (b) the procedure of GN update

3 | PROBLEM SPECIFICATION

Routing optimization is usually modelled as a multi-commodity flow problem [30], which is simply a network flow problem with multiple demands between different sources and sink nodes. It consists of a flow network $G(V, E)$ where each edge $(u, v) \in E$ has a capacity $c(u, v)$. Then, we wish to place commodities $K_i = (s_i, t_i, d_i)$ on the network under the following constraints:

$$\begin{cases} \forall (u, v) \in E, \sum_{i=1}^k f_i(u, v) \cdots d_i \leq c(u, v) & \# \text{not exceed capacity} \\ \sum_{w \in V} f_i(u, w) - \sum_{w \in V} f_i(w, u) = 0 & \# \text{must be conserved} \\ \sum_{w \in V} f_i(s_i, w) - \sum_{w \in V} f_i(w, s_i) = 1 & \# \text{exit its source} \\ \sum_{w \in V} f_i(w, t_i) - \sum_{w \in V} f_i(t_i, w) = 1 & \# \text{absorbed at the sink} \end{cases} \quad (7)$$

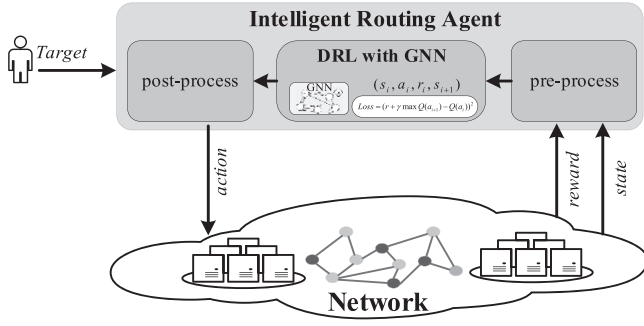


FIGURE 2 Framework of intelligent routing in network

where $f_i(u, w)$ represents the number of flows from u to w on node i .

As long as we allow fractional flows, we can optimize the routing under a given utility function using linear programming in polynomial time for a specified set of commodities. However, we do not have prior knowledge of the traffic demands on a network, so the routing would always lag the commodities for which it was optimized.

Therefore, we should take the commonly used network state instead of traffic demands as the input of route optimization. The network status includes link congestion, port throughput, queue waiting, and so forth, which could be collected easily through OpenFlow or INT protocol. Although the network status changes dynamically, it is usually continuous and may be periodic, which means that the historical status data in $(t-t_b, t)$ is useful for optimizing routing in the future of $(t, t + \Delta t)$. The time interval Δt is not fixed and is adjusted as needed according to the change range of the network status.

But the optimal routing policy cannot be obtained from the network status, because it does not cover (or cannot derive) the traffic demands. In this paper, the expected result is an approximate optimal solution, or better than obvious routing [31]. As shown in Figure 2, we explore a DRL-based intelligent routing agent, inputting the above network state as the state (denoted as s), outputting flow-table entities as the action (denoted as a), and obtaining the effect feedback reward (denoted as r). The agent gradually optimizes the routing policy to approach the target through repeated interactions with the network environment, such as $(s_i, a_i, r_i, s_{i+1}, \text{etc.})$. And the agent should have the ability of generalized reasoning, that is, the learned routing optimization policy is also suitable for networks of different topologies without retraining.

4 | GNN-BASED DRL AGENT DESIGN

The focus of the agent design is how it provides intelligent and efficient routing policies. In order to achieve the functions described in Section 3, the agent needs to pre-process the data collected from the network to the standard state and reward, train the DRL algorithm with generalization ability, and post-process the action to the detailed routing path.

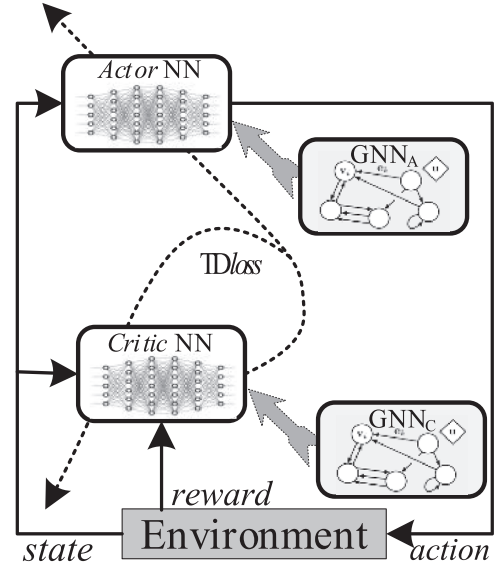


FIGURE 3 Improve PPO with GNN

4.1 | GNN-based DRL algorithm

We introduced PPO and GNN in Section 2, which are state-of-the-art AI technologies. Here, we are trying to combine the two together to obtain a DRL algorithm with generalization ability, that is, to explore a GNN-based DRL algorithm. Conventional PPO uses MLP, CNN or LSTM as the neural network of Actor and Critic, which lack flexibility as their input and output are both fixed. As shown in Figure 3, we will replace the neural network in PPO with GNN to improve its generalization ability, in which GNN models the changing topology of networks (described in Sections 4.2, 4.3, and 4.4).

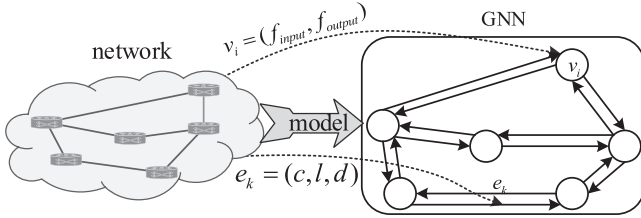
GNN_A and GNN_C , respectively, represent the neural networks of Actor and Critic in the improved PPO, which should have the same graph structure in order to model the same network. At the same time, GNN_A and GNN_C are related by formula (5), including: ① train GNN_C according to the TD loss in formula (6), and then calculate its partial derivative $\frac{\partial \mathcal{L}^\theta(s, a, \omega)}{\partial a}$; ② calculate the vector product of the above partial derivative and $\partial a / \partial \theta$, used for gradient optimization of GNN_A parameters. Specifically, we describe a pseudo-code of GNN-based PPO in algorithm 1, where KL represents the divergence between two distributions [20].

4.2 | State

The state is the data collected from networks, including link congestion, port throughput, queue waiting, and so forth. Since the dimension of the state is not fixed, we need to explain how to map these irregular data into the input of GNN-based DRL. First, we model the GNN (GNN_A and GNN_C) with the same topology as its corresponding network. As shown in Figure 4, the nodes in the network correspond to the nodes in the GNN one-to-one, and each link in the network

ALGORITHM 1 GNN-based PPO algorithm

1. **for** $i \in \{1, \dots, N\}$ **do**
2. **Run policy** π_θ **for** T , **collecting** (s_i, a_i, r_i)
3. **Estimate** $\hat{A}_i = \sum_{t' > i} \gamma^{t'-i} r_{t'} - V_\phi(s_i)$
4. $\pi_{old} = \pi_\theta$
5. **for** $j \in \{1, \dots, M\}$ **do**
6. $J_{PPO}(\theta) = \sum_{i=1}^T \frac{\pi_\theta}{\pi_{old}} \hat{A}_i - \lambda \text{KL}[\pi_{old} | \pi_\theta]$
7. **Update** θ in GNN_A **with** $J_{PPO}(\theta)$
8. **for** $j \in \{1, \dots, B\}$ **do**
9. $L_{BL}(\phi) = - \sum_{i=1}^T (\hat{A}_i)^2$
10. **Update** ϕ in GNN_C **with** $L_{BL}(\phi)$
11. **if** $\text{KL}[\pi_{old} | \pi_\theta] > \beta_j \text{KL}_{\text{arg et}}$ **then**
12. $\lambda \leftarrow \lambda \dots \alpha$
13. **else if** $\text{KL}[\pi_{old} | \pi_\theta] < \beta_j \text{KL}_{\text{arg et}}$ **then**
14. $\lambda \leftarrow \lambda / \alpha$

**FIGURE 4** Model GNN from the network

(distinguishing between uplink and downlink) corresponds to a directed edge in the GNN. Then, we pre-process the statistical data obtained from the switch into the attribute v_i of the GNN node, for example, the average outgoing traffic f_{input} Mbps and incoming traffic f_{output} Mbps. And we pre-process the link-related data into the attribute e_k of the edge in the GNN, for example, the transmission rate c Mbps, the average length l of the queue in its connected port, and the packet loss rate d . Here, we should pay attention to the change and the unchanged in GNN: (1) the topology is changed to adapt to different networks; (2) the dimensions of the attribute v_i and e_k remain unchanged, which requires pre-processing to unify.

Furthermore, the above-mentioned data may be a fixed-length list of some recent history, not only the statistical value in the latest time period. Expanding the input state data in the time dimension is usually helpful for DRL to better learn the laws of the environment, but there may be a threshold in effect, so the calculation time needs to be considered comprehensively.

4.3 | Action

The action is the data related to calculating the routing path. As mentioned in Section 3, the optimal routing policy needs to split any flow at any node in any proportion, but this is not practical

for intelligent routing. The reasons include: (1) it would require the DRL to output $|V| \times (|V| - 1) \times |E|$ separate values. Unfortunately, this size of action space is too big to enable successful learning; (2) Compared with the commonly used OSPF protocol, it will increase flow entries in the switch many times, which is not conducive to the scalability of the network. Therefore, we only pursue the sub-optimal effect of the action output by the DRL, which is a compromise between performance and complexity.

In this paper, we use an improved WCMP (weight-cost multi-path routing) to select p paths for each flow according to the weight of the link. Adjusting routing based on link weight is more fine-grained and is usually better than directly selecting among multiple routing paths. Therefore, the link weight is the direct output of the GNN-based DRL algorithm. First, the GNN_A in the DRL with the input of state (v_i, e_k) will output the final attributes of the edge after multiple rounds of message passing, which is used as the weight of its corresponding link. The final attribute of the node has nothing to do with the action, and is not used for parameter updates in PPO. Then, we calculate p shortest paths for each flow, and split the traffic according to the proportion of *softmax*, as follows,

$$\text{softmax}(\text{path}_i) = \frac{e^{-w_i}}{\sum_{j \leq p} e^{-w_j}} \quad (8)$$

where w_j represents the sum of link weight of the j th shortest path. Finally, we adjust the value of p to avoid waste of routing resources for a small portion of traffic, that is,

$$\begin{cases} \text{Max}(p) \\ s.t. \frac{e^{-w_{p+1}}}{\sum_{j \leq p+1} e^{-w_j}} \leq 0.2 \end{cases} \quad (9)$$

Like the state, since the dimension of the action can change with the change of the network topology, the GNN-based DRL has the possibility to optimize routing in general.

4.4 | Reward

The reward is also the data collected from networks, which is the feedback to the action taken by the DRL agent. Unlike the state and action mentioned above, the reward should be a scalar, not a multi-dimensional tensor. Therefore, we need to convert these data into a single value based on the target of routing optimization. The setting of reward has nothing to do with the generalization ability of GNN, and the methods in existing research can be used for reference. For example, [1] models the reward as the ratio between the average throughputs of two consecutive time steps, that is,

$$r_t = \frac{\sum_{f \in F_d^t} \text{Thput}_f^t}{\sum_{f \in F_d^{t-1}} \text{Thput}_f^{t-1}} \quad (10)$$

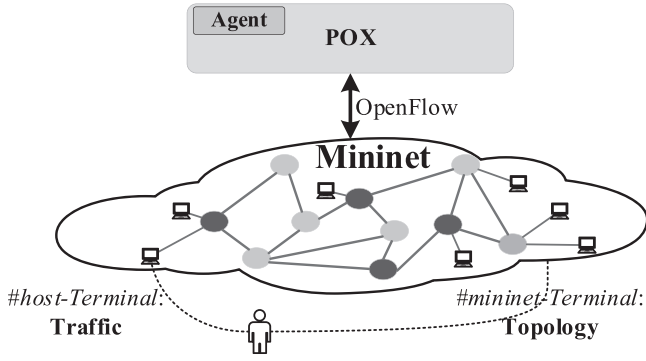


FIGURE 5 Experimental environment

where F represents a set of traffic flows in the network and T_{put} represents the throughput of one flow. Or [18] models the reward as the effect of reducing link congestion, which gives smaller increases in the ratio closer to the optimal congestion a higher relative reward.

The administrator can freely set the evaluation function of reward for different goals, so here we will not instantiate it. In the next experiments, we define reward based on specific application scenarios.

5 | EVALUATION

5.1 | Setup

In order to emphasize the generalization and practicality of our proposed intelligent routing method, we deploy it in a real SDN network for experimentation rather than simulation. As shown in Figure 5, the intelligent routing is implemented as a northbound application *Agent* on the controller POX, which is responsible for calculating routing and generating flow entries. We use Mininet to create the data plane, in which the traffic demand is input through the terminal of the virtual host, and the network topology is managed through the terminal of Mininet. In addition, we modified the Open vSwitch in Mininet according to the open project to support multipath routing. POX periodically interacts with the switch through the OpenFlow protocol, collecting state/reward data in the network and then downloading flow entries related to the action. Here we define reward as:

$$r_t = -\frac{1}{|V|(|V|-1)} \sum_{i,j \in V} delay_{(i,j)} \quad (11)$$

which means reducing the average delay of the network as the target of the DRL Agent.

Specifically, we use the PPO from the stable-baselines library and the GNN from GraphNets library to implement the GNN-based DRL algorithm. In the GNN, we select MLP with the size of (32, 64, 64, 32) for links' and nodes' update functions. In every forward propagation of GNN, we execute $T=12$ message passing steps using batches of 32 samples. The optimizer used is

an *AdamOptimizer*. As for the hyperparameters in the DRL, we use a learning rate of 2.5×10^{-4} , and a momentum of 0. For the ϵ -greedy exploration strategy, we start with $\epsilon = 0.01$, and this value is maintained during 50 training iterations. Afterwards, ϵ decays with 0.95. To stabilize the learning process, we re-train the weights of the GNN model from the experience buffer. The experience replay buffer size is set to store 1,00,000 samples, and it is implemented as a prioritized buffer: once it is full, the experience with low priority will be discard.

In the data plane, we select three typical network topologies [32], including GBN, NSFNet, and GEANT2. During the test, we can replace the entire network by restarting Mininet, or make minor changes to a network by adding or deleting some links or nodes. On the other hand, the traffic demand in the network should have some sort of regularity which we can exploit to predict a routing strategy for the next timestep. And these scenarios make sense as temporal regularity is often seen in large networks such as those of ISPs [18], and middle networks such as campus LAN [33]. In our experiments, we use an averaging sequence as the kind of regularity, which is generated by taking a sequence of demand matrices and averaging over the last n when outputting the next demand. And the averaging sequence come in *gravity* demand, that is,

$$D_{ij} = \sum_{(i,v) \in E} c(i,v) \times \sum_{(u,j) \in E} c(u,j) \quad (12)$$

where c is the edge's capacity. D_{ij} is a deterministic manner taking into account the structure of the network. To add variance allowing us to build sequences of different demands, we regularly average the D_{ij} according to the traffic demand trend of Stanford University in [33] with a period of 24 min (we shorten the 24-h period to 24 min).

5.2 | Result and analysis

5.2.1 | Training

First, we verify the effectiveness of the proposed intelligent routing method, that is, whether the DRL Agent can reduce the average delay of the network during training. Here, we choose NSFNet as the network in the data plane and do not change its topology. We set the Agent to sample data from the network every 30 s, recalculate the routing paths and download flow entries. At the same time, based on the regularity of traffic, we regard 24 min as an episode duration of DRL, which is also a basic unit of statistical analysis. After the hyperparameters of GNN are optimized, we only do each experiment once, but the data for each point in the following figures is the average of 48 statistics.

The training result is shown in Figure 6; the ordinate axis represents the average of all sampling network delays in an episode, so that the different delays caused by different traffic demands at different times can be averaged. We can observe that the average delay of the network is gradually decreasing during training and the learning remains stable after 80 episodes (32 h) approximately. After stabilization, the network delay is reduced by about

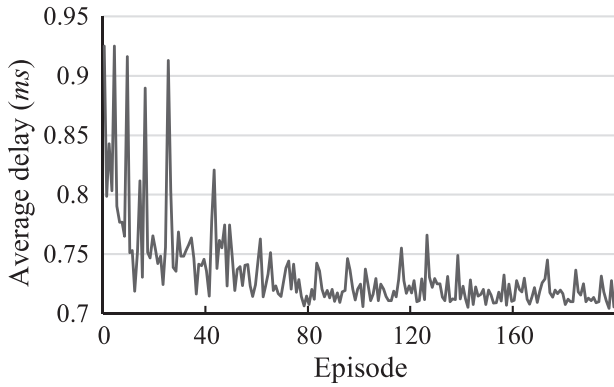


FIGURE 6 Average delay evolution during training

11%, proving that the intelligent routing method is effective in SDN networks.

5.2.2 | Generalization

Then, we verify the generalization ability of the GNN-based DRL Agent, that is, whether the well-trained DRL algorithm can still function when the network topology changes. Therefore, we will change the topology of the NSFNet network after the Agent has been trained for 100 episodes, at which point it has reached stability. There are two types of network topology changes, including: (1) Partial changes, randomly add/delete some links or nodes through Mininet terminal commands; (2) Overall changes, replace the network with GBN or GEANT2 by restarting Mininet. Like the above training, we count the average delay in each episode, regardless of whether the network topology changes. At the same time, we have also compared with DROM [34] in the experiment of partial changes, although it can only work in fixed-topology networks. When the original link is deleted, DROM still calculates its weight, but it is not used. When a new link is added, DROM cannot calculate its weight, so we set it to the default value 1. In the experiment of overall changes, it is meaningless to use DROM according to the above method, so we will no longer compare with it, and only analyse it qualitatively from the average delay evolution of its own results.

The experimental results are shown in Figure 7, where (a) corresponds to partial changes: we can observe that after stabilization (Train), even though the network topology has undergone small changes many times, the DRL Agent (GRL) can maintain the original optimization effect, that is, the same trend of average delay evolution as in Figure 6. But DROM is different. Once the network topology changes, the network delay will increase sharply, which means it cannot continue to optimize routing. And (b) corresponds to the overall changes: after the network is replaced with GBN or GEANT2, the DRL Agent can still optimize routing to a lower network delay (compared to the average delay 0.874 ms of the above DROM) without being retrained, which can be proved that the fluctuation of their average delay is roughly consistent with that of NSFNet. In summary, the GNN-based DRL algorithm we pro-

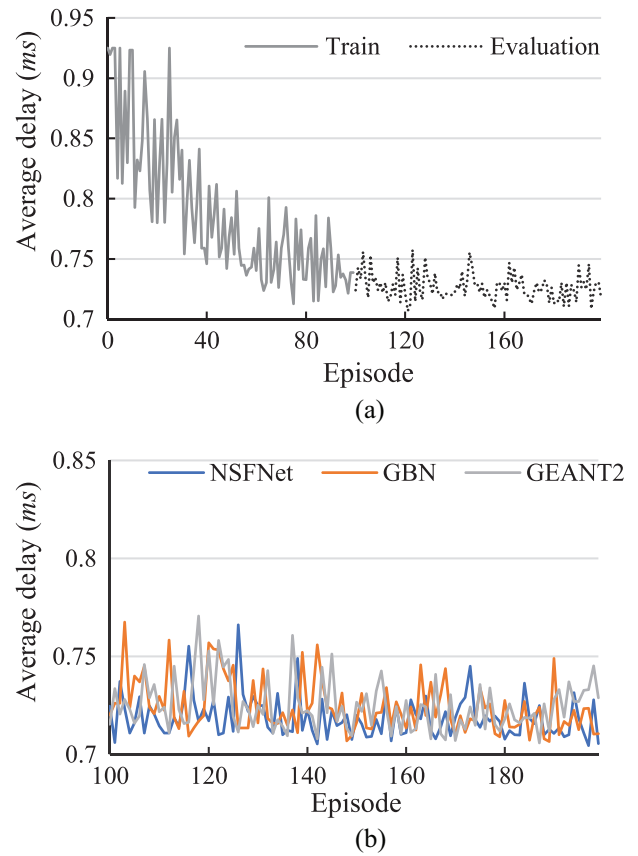


FIGURE 7 The GRL agent reacts to topology changes. (a) Average delay evolution after partial changes, (b) average delay evolution after overall changes

posed can optimize routing in unseen networks, that is, it has important generalization capabilities.

5.2.3 | Comparison

Finally, we test the optimization effect of the GNN-based DRL Agent (denoted as GRL) after stability and compare with four other conventional routing policies. They are (1) the shortest route based on the number of hops, denoted as hop; (2) the route of optimal load balancing (minimizing the maximum link utilization) solved by linear programming, denoted as LP. Here, the minimum average delay is not used as the objective function of LP because it cannot be modelled as a mathematical expression; (3) a recent deep reinforcement learning approaches for routing, such as the aforementioned DROM. (4) A recent generalizable data-driven routing method with GNN + DRL [18], abbreviated as DDR. In addition, we record the total number of flow entries generated under each routing policy. Among them, the group entries used in multipath routing are equivalent to multiple flow entries with the same function.

The experimental results are shown in Figure 8, where the average delay is calculated during the 100 episodes of Mininet running. We can observe that LP wastes a huge flow-table space but has a delay that is worse than hop, so this complicated routing policy is not practical in the network. The average delay of the DRL algorithm is the smallest, which is reduced by 16.1%

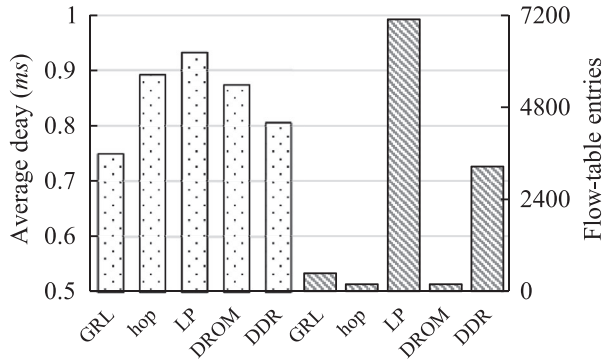


FIGURE 8 Comparison of three routing strategies

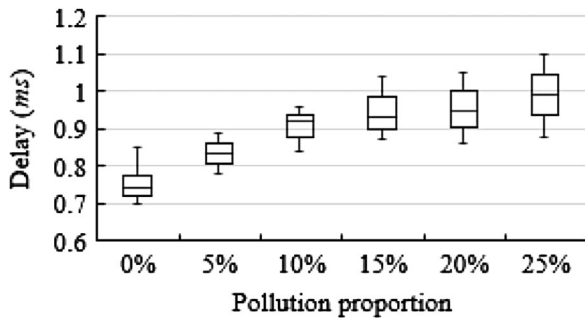


FIGURE 9 The robustness of the GRL

relative to hop, 19.6% relative to LP, and 14.3% relative to DROM. But the number of flow entries generated by the DRL algorithm is increased by 1.01 times relative to hop, which is acceptable for the Open vSwitch in Mininet. Compared with the recent generalizable data-driven routing method DDR, our proposed intelligent routing algorithm GRL can reduce the delay by 7% and save 85.6% of the flow-table space. Therefore, the GRL algorithm can again compromise between network delay and flow entry by adjusting the constraint ratio in formula (7) (e.g. '0.2'), which may be a meaningful exploration.

5.2.4 | Robustness

The quality of the training data has a significant impact on the performance of an AI algorithm, that is, polluted data may cause the algorithm to not converge or optimize. Therefore, the robustness of GRL is also a metric we evaluate. Meta-Attack is a common poisoning attack method for GNNs [35], which can significantly reduce the inference accuracy by only small modifications to the feature data on nodes or structure. Here, we choose NSFNet as the network in the data plane and implement Meta-Attack on its nodes while keeping the NSFNet topology unchanged. As for the missing edges in the GNN topology, which are the input for the computation of the route, we use the value from its previous interaction (may be the initial value). Then, the polluted data is used as the *state* of GRL to test its robustness against poisoning attacks.

The experimental results are shown in Figure 9, where the X-axis is the proportion of polluted data, reflecting the degree of Meta-Attack; the Y-axis is the average delay of the network,

which is the statistical data within 100–200 episodes, and is analysed in the form of a box plot. We can observe that as the proportion of data polluted increases, the optimization effect of GRL gradually decreases. When the pollution proportion reaches 15%, GRL approximates the performance of the conventional algorithm hop (described in Section 5.2.3). When it reaches 25%, GRL approximates random routing and completely loses the optimization effect. It is more meaningful that when the pollution proportion is less than 10%, GRL can still have a good optimization effect, which may be related to the 'Experience Replay Buffer' in the DRL algorithm.

Although GRL has certain robustness properties, we should preprocess the collected data first to avoid some pollutants affecting the optimization effect of the intelligent routing algorithm.

6 | CONCLUSION

We contributed in this paper a novel approach for practical intelligent routing method using DRL with GNN, which could be easily implemented as a northbound application on the SDN controller to optimize traffic. Our method can not only output continuous control actions for routing optimization but also learn from some network and generalize to other unseen ones.

In order to emphasize the generalization and practicality of our proposed intelligent routing method, we deploy it in a real SDN network for experimentation rather than simulation. The results show that the method can keep on optimizing the routing of traffic in networks of different topologies after the training is stable. And compared with hop-based OSPF and the optimal load-balancing algorithm, it reduces network delay by 4.3% and 5.7%, respectively.

As the state data of the DRL Agent obtained through the OpenFlow is difficult to sketch the network, our further work is to extend it through the INT protocol in order to achieve better routing.

ACKNOWLEDGEMENTS

This work was supported by the National Natural Science Foundation of China (No. 62002382, No. 62072416); The Project of Science and Technology in Henan Province (No. 222102210175, No. 222102210111); Postgraduate Education Reform and Quality Improvement Project of Henan Province (No. YJS2022AL035).

CONFLICT OF INTEREST

The authors have declared no conflict of interest.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available on request from the corresponding author.

ORCID

Bo Yuan  <https://orcid.org/0000-0002-8401-6709>

REFERENCES

- Chen, L., Lingys, J., Chen, K., Liu, F.: AuTO: scaling deep reinforcement learning for datacenter-scale automatic traffic optimization.

- In: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, pp. 191–205. ACM, New York, USA (2018)
2. Bai, W., Chen, L., Chen, K., Han, D., Tian, C., Wang, H.: PIAS: practical information-agnostic flow scheduling for commodity data centers. *IEEE/ACM Trans. Netw.* 25(4), 1954–1967 (2017)
3. Alizadeh, M., Edsall, T., Dharmapurikar, S., Vaidyanathan, R., Chu, K., Fingerhut, A., Lam, V.T., Matus, F., Pan, R., Yadav, N., Varghese, G.: CONGA: Distributed congestion-aware load balancing for datacenters. In: Proceedings of the 2014 ACM conference on SIGCOMM, Chicago, Illinois, USA, 17–22 August 2014, pp. 503–514. Association for Computing Machinery, New York, USA (2014)
4. He, K., Rozner, E., Agarwal, K., Felten, W., Carter, J., Akella, A.: Presto: edge-based load balancing for fast datacenter networks. In: Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, pp. 465–478. Association for Computing Machinery, New York, USA (2015)
5. Zhao, Y., Chen, K., Bai, W., Yu, M., Tian, C., Geng, Y., Zhang, Y., Li, D., Wang, S.: Rapier: integrating routing and scheduling for coflow-aware data center networks. In: Proceedings of the 2015 IEEE Conference on Computer Communications, pp. 424–432. IEEE, Hong Kong, P. R. China (2015)
6. Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., Zolla, J., Hözl, U., Stuart, S., Vahdat, A.: B4: experience with a globally-deployed software defined wan. In: Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM, pp. 3–14. Association for Computing Machinery, New York, USA (2013)
7. Alizadeh, M., Yang, S., Sharif, M., Katti, S., McKeown, N., Prabhakar, B., Shenker, S.: pFabric: minimal near-optimal datacenter transport. In: Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM, pp. 435–446. Association for Computing Machinery, New York, USA (2013)
8. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. *ACM* 60(6), 84–90 (2017)
9. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of Go with deep neural networks and tree search. *Nature* 529(7587), 484–489 (2016)
10. Xiao, S., He, D., Gong, Z.: Deep-Q: traffic-driven QoS inference using deep generative network. In: Proceedings of the 2018 Workshop on Network Meets AI & ML, pp. 67–73. Association for Computing Machinery, New York, USA (2018)
11. Salman, S., Streiffer, C., Chen, H., Benson, T., Kadav, A.: DeepConf: automating data center network topologies management with machine learning. In: Proceedings of the 2018 Workshop on Network Meets AI & ML, pp. 8–14. Association for Computing Machinery, New York, USA (2018)
12. Mestres, A., Rodriguez-Natal, A., Carner, J., Barlet-Ros, P., Alarcón, E., Solé, M., Muntés-Mulero, V., Meyer, D., Barkai, S., Hibbett, M.J., Estrada, G., Ma'ruf, K., Coras, F., Ermagan, V., Latapie, H., Cassar, C., Evans, J., Maino, F., Walrand, J., Cabellos, A.: Knowledge-defined networking. *ACM SIGCOMM Comput. Commun. Rev.* 47(3), 2–10 (2017)
13. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., Hassabis, D.: Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv:1712.01815* (2017)
14. Stampa, G., Arias, M., Sanchez-Charles, D., Muntés-Mulero, V., Cabellos, A.: A deep-reinforcement learning approach for software-defined network routing optimization. *arXiv:1709.07080* (2017)
15. Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M.: Deterministic policy gradient algorithms. In: Proceedings of the 31st International Conference on Machine Learning, pp. 387–395. PMLR, Beijing, P. R. China (2014)
16. Almasan, P., Suárez-Varela, J., Badia-Sampera, A., Rusek, K., Barlet-Ros, P., Cabellos-Aparicio, A.: Deep reinforcement learning meets graph neural networks: exploring a routing optimization use case. *arXiv:1910.07421* (2019)
17. Battaglia, P.W., Hamrick, J.B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G., Vaswani, A., Allen, K., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., Pascanu, R.: Relational inductive biases, deep learning, and graph networks. *arXiv:1806.01261* (2018)
18. Hope, O.D.: Generalisable data-driven routing using Deep RL with GNNs. M.S. thesis, Dept. Computer. Tec., University of Cambridge (2020)
19. Wang, H., Xu, H., Qian, C., Ge, J., Liu, J., Huang, H.: PrePass: load balancing with data plane resource constraints using commodity SDN switches. *Comput. Netw.* 178(4), 107339 (2020)
20. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. *arXiv:1707.06347*, (2017)
21. Kakade, S.M.: A natural policy gradient. *Adv. Neural Inf. Process. Syst.* 14, 1531–1538 (2001)
22. Peters, J., Schaal, S.: Natural actor-critic. *Neurocomputing* 71(7–9), 1180–1190 (2008)
23. Lapan, M.: Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more. Packt Publishing Ltd, Birmingham, UK (2018)
24. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry. *Proc. 34th Int. Conf. Machine Learn.* 70, 1263–1272 (2017)
25. Wang, X., Girshick, R., Gupta, A., He, K.: Non-local neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 7794–7803. IEEE, Salt Lake City, USA (2018)
26. Sanchez-Gonzalez, A., Heess, N., Springenberg, J.T., Merel, J., Riedmiller, M., Hadsell, R., Battaglia, P.: Graph networks as learnable physics engines for inference and control. *Proc. 35th Int. Conf. Machine Learn.* 80, 4470–4479 (2018)
27. Nowak, A., Villar, S., Bandeira, A.S., Bruna, J.: A note on learning algorithms for quadratic assignment with graph neural networks. *arXiv:1706.07450* (2017)
28. Selsam, D., Lamm, M., Bünz, B., Liang, P., de Moura, L., Dill, D.L.: Learning a SAT solver from single-bit supervision. *arXiv:1802.03685* (2018)
29. Yoon, K., Liao, R., Xiong, Y., Zhang, L., Fetaya, E., Urtasun, R., Zemel, R., Pitkow, X.: Inference in probabilistic graphical models by graph neural networks. In: 2019 53rd Asilomar Conference on Signals, Systems, and Computers, pp. 868–875. IEEE, Pacific Grove, CA, USA (2019)
30. Clausen, J.V., Lusby, R., Ropke, S.: Routing problems, multi-commodity fixed-charge network design and variable splitting. *ROUTE 2018: International Workshop on Vehicle Routing, Intermodal Transportation and Related Areas*, pp. 103–106. Sneckersten, Denmark (2018)
31. Kumar, P., Yuan, Y., Yu, C., Foster, N., Kleinberg, R., Lapukhov, P., Lim, C.L., Soule, R.: Semi-oblivious traffic engineering: the road not taken. In: Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation, Renton, WA, USA, 9–11 April 2018, pp. 157–170. USENIX Association, Renton, USA (2018)
32. Suárez-Varela, J., Mestres, A.: Knowledge-defined networking training datasets. <http://knowledgeDEFINEDnetworking.org/>. Accessed 2019
33. Shalimov, A., Zuikov, D., Zimarina, D., Pashkov, V., Smeliansky, R.: Advanced study of SDN/OpenFlow controllers. In: Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia, pp. 1–6. ACM, New York, USA (2013)
34. Yu, C., Lan, J., Guo, Z., Hu, Y.: DROM: optimizing the routing in software-defined networks with deep reinforcement learning. *IEEE Access* 6, 64533–64539 (2018)
35. Jin, W., Li, Y., Xu, H., et al.: Adversarial attacks and defenses on graphs. *ACM SIGKDD Explorations Newsletter* 22(2), 19–34 (2021)

How to cite this article: Huang, W., Yuan, B., Wang, S., Zhang, J., Li, J., Zhang, X.: A generic intelligent routing method using deep reinforcement learning with graph neural networks. *IET Commun.* 16, 2343–2351 (2022). <https://doi.org/10.1049/cmu2.12487>