

## Article

# A Routing Optimization Method for Software-Defined Optical Transport Networks Based on Ensembles and Reinforcement Learning

Junyan Chen <sup>1,2</sup> , Wei Xiao <sup>1</sup>, Xinmei Li <sup>1</sup>, Yang Zheng <sup>3,\*</sup>, Xuefeng Huang <sup>1</sup>, Danli Huang <sup>1</sup> and Min Wang <sup>1</sup>

<sup>1</sup> School of Computer Science and Information Security, Guilin University of Electronic Technology, Guilin 541004, China

<sup>2</sup> School of Computer Science and Engineering, Northeastern University, Shenyang 110819, China

<sup>3</sup> Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China

\* Correspondence: yang.zheng@ia.ac.cn

**Abstract:** Optical transport networks (OTNs) are widely used in backbone- and metro-area transmission networks to increase network transmission capacity. In the OTN, it is particularly crucial to rationally allocate routes and maximize network capacities. By employing deep reinforcement learning (DRL)- and software-defined networking (SDN)-based solutions, the capacity of optical networks can be effectively increased. However, because most DRL-based routing optimization methods have low sample usage and difficulty in coping with sudden network connectivity changes, converging in software-defined OTN scenarios is challenging. Additionally, the generalization ability of these methods is weak. This paper proposes an ensembles- and message-passing neural-network-based Deep Q-Network (EMDQN) method for optical network routing optimization to address this problem. To effectively explore the environment and improve agent performance, the multiple EMDQN agents select actions based on the highest upper-confidence bounds. Furthermore, the EMDQN agent captures the network's spatial feature information using a message passing neural network (MPNN)-based DRL policy network, which enables the DRL agent to have generalization capability. The experimental results show that the EMDQN algorithm proposed in this paper performs better in terms of convergence. EMDQN effectively improves the throughput rate and link utilization of optical networks and has better generalization capabilities.

**Keywords:** optical transport network; software-defined networking; deep Q-network; message-passing neural network; ensemble learning



**Citation:** Chen, J.; Xiao, W.; Li, X.; Zheng, Y.; Huang, X.; Huang, D.; Wang, M. A Routing Optimization Method for Software-Defined Optical Transport Networks Based on Ensembles and Reinforcement Learning. *Sensors* **2022**, *22*, 8139. <https://doi.org/10.3390/s22218139>

Academic Editors: Chien Aun Chan, Ming Yan and Chunguo Li

Received: 11 October 2022

Accepted: 20 October 2022

Published: 24 October 2022

**Publisher's Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Copyright:** © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

The optical transport network (OTN) is a transport network that enables the transmission, multiplexing, route selection, and monitoring of service signals in an optical domain, ensuring its performance index and survivability. The OTN can support the transparent transmission of customer signals, high-bandwidth multiplexing, and configuration. It also provides end-to-end connectivity and networking capabilities. With the rapid development of network communication technology, the demand for OTN networks has increased significantly in terms of the scale of information volume, demand complexity, and dynamic spatio-temporal distribution. Unlike traditional networks, the OTN can meet more network requirements due to its suitable transmission medium, which has a high transmission speed, more data transmission, and a long transmission distance.

Traditional routing design schemes manually model network demand characteristics and design routing policies in a focused way. The traditional routing protocol is designed for wired networks, with a fixed bandwidth allocation pattern and low bandwidth utilization. It cannot provide differentiated services based on the level of assistance, nor can it cope with the rapid changes in topology and link quality standards in optical network environments.

Additionally, because OTN demand has complex spatio-temporal distribution fluctuations, the optimization problem of its routing is an NP-hard problem [1]. In this case, traditional network routing design schemes do not apply to the OTN.

With the development of new network architectures, such as the software-defined networking (SDN) and the maturation of deep reinforcement learning (DRL) techniques in recent years, software-defined optical transport networks (SD-OTNs) based on the SDN are gaining popularity in the industry. Recent studies have used the DRL to address SDN-related problems, such as QoS-aware secure routing for the SDN-IoT [2], SDN routing optimization problems [3], and the SDN demand control [4]. However, due to DRL agents' lack of generalization capabilities, they do not achieve good results in new network topologies. Thus, DRL agents cannot make correct routing decisions when presented with unexplored network scenarios during the training phase. The main reason behind this phenomenon is that graphs essentially represent computer networks. In addition, traditional DRL algorithms use typical neural network (NN) architectures (e.g., fully connected convolutional neural networks), which are unsuitable for modeling information about graph structures. Due to the computational effort and high time complexity of the routing optimization problem, traditional DRL algorithms are challenging for the DRL agent to converge quickly when addressing the network routing optimization problem. Additionally, OTN network problems are incredibly complex and have high trial-and-error costs, making it difficult to implement DRL algorithms in real optical networks.

This paper proposes an ensembles- and message-passing neural-network-based deep Q-network (EMDQN) method to solve the SD-OTN routing decision problem. The message-passing neural network (MPNN) is a deep learning (DL) method based on a graph structure [5]. The MPNN contributes to learning the relationship between graph elements and their rules. In this paper, the MPNN is used to capture information about the relationship between the demand on links and network topology, which can improve the model's generalization ability. Despite computationally complex network problems, ensemble learning has a unique advantage that can increase sample utilization. We reweigh the sample transitions based on the uncertainty estimates of ensemble learning. This method can improve the signal-to-noise ratio during Q-network updates, and stabilize the learning process of the EMDQN agent, which helps the deep Q-network (DQN) [6] operate stably in OTN networks.

The main contributions of this paper are as follows:

1. We propose an SD-OTN routing optimization algorithm based on the reinforcement learning model of the EMDQN. To effectively improve the extrapolation capability of DRL decision-makers, we design a more refined state representation and a limited set of actions.
2. We use the MPNN algorithm instead of the traditional DQN's policy networks, which can capture the relationship between links and network topology demand and improve the DRL decision-maker performance and generalization capability. Additionally, we exploit the advantages of efficient exploration through ensemble learning to explore the environment in parallel and improve convergence performance.
3. We design practical comparison experiments to verify the superior performance of the EMDQN model.

The rest of this paper is structured as follows. In Section 2, this paper discusses research related to the proposed solution for the network problem. Section 3 describes the software-defined network system architecture and the OTN optimization scenarios and tasks. In Section 4, this paper describes the design of DRL-based routing optimization decisions. In Section 5, this paper presents an extensive evaluation of DRL-based solutions in some realistic OTN scenarios. Finally, in Section 6, we present our conclusion and directions for future work.

## 2. Related Research

Traditional routing optimization schemes are usually based on the OSPF (open shortest path first) [7] or ECMP (equal-cost multipath routing) [8]. The OSPF protocol routes all flow requests individually to the shortest path. The ECMP protocol increases transmission bandwidth using multiple links simultaneously. However, these approaches, based on fixed forwarding rules, are prone to link congestion and cannot meet the demand of exponential traffic growth. Recently, most heuristic algorithm-based approaches have been built under the architecture of the SDN. The authors in [9] proposed a heuristic ant-colony-based dynamic layout algorithm for SDNs with multiple controllers, which can effectively reduce controller-to-switch and controller-to-controller communication delays caused by link failures. The authors in [10] applied a random-based heuristic method called the alienated ant algorithm, which forces ants to spread out across all available paths while searching for food rather than converging on a single path. The authors in [11] analytically extract historical user data through a semi-supervised clustering algorithm for efficient data classification, analysis, and feature extraction. Subsequently, they used a supervised classification algorithm to predict the flow of service demand. The authors in [12] proposed a heuristic algorithm-based solution for DWDM-based OTN network planning. The authors in [13] proposed a least-cost tree heuristic algorithm to solve the OTN path-sharing and load-balancing problem. However, because of a lack of historical experience in data learning, heuristic algorithms can only build models for specific problems. When the network changes, it is difficult to determine the network parameters and there is limited scalability to guarantee service quality. Furthermore, because of the tremendous computational effort and high computational complexity of these methods, heuristic algorithms do not perform well on OTN networks.

With SDN's maturity and large-scale commercialization, the SD-OTN based on the SDN is becoming increasingly popular in the industry. SD-OTN adapts the reconfigurable optical add-drop multiplexer (ROADM) nodes through the southbound interface protocol and establishes a unified resource and service model. The SD-OTN controller can realize topology and network status data collection, routing policy distribution, and network monitoring. Therefore, many researchers deploy artificial intelligence algorithms in the controller. Deep learning, with its powerful learning algorithms and excellent performance advantages, has gradually been applied to the SDN. To solve the SDN load-balancing problem, Chen et al. [14] used the long short-term memory (LSTM) to predict the network traffic in the SDN application plane. The authors in [15] proposed a weighted Markov prediction model based on mobile user classification to optimize network resources and reduce network congestion. The authors in [16] proposed an intrusion detection system based on SDN and deep learning, reducing the burden of security configuration files on network devices. However, deep learning requires many datasets for training and has poor generalization abilities due to its inability to interact with the environment. These factors make it difficult to optimize the performance of dynamic networks. Compared with deep learning, reinforcement learning uses online learning for model training, changing agent behaviors through continuous exploration, learning, and experimentation to obtain the best return. Therefore, reinforcement learning does not require the model to be trained in advance. It can change its action according to the environment and reward feedback. The authors in [17] designed a Q-learning-based localization-free routing for underwater sensor networks. The authors in [18] proposed a deep Q-routing algorithm to compute the path of any source-destination pair request using a deep Q-network with prioritized experience replay. The authors in [19] proposed traction control ideas to solve the routing problem. The authors in [20] proposed a routing optimization algorithm based on the proximal policy optimization (PPO) model in reinforcement learning. The authors in [21] discussed a solution for automatic routing in the OTN using DRL. Although the studies described above have been successful for the SDN demand-routing optimization problem, they do not perform as well in new topologies because they do not consider the model's generalization capability.

The traditional DRL algorithms use a typical neural network (NN) as the policy network. The NN can extract and filter the features of the input information and data layer by layer to finally obtain the results of tasks, such as classification and prediction. However, as research advances, conventional neural networks are unable to solve all network routing problems and will struggle to handle non-Euclidean-structured graph data. Therefore, we need to optimize the traditional reinforcement learning algorithm to improve its ability to extract the information features of the sample. Off-policy reinforcement learning (Off-policy RL) algorithms significantly improve sample utilization by reusing past experiences. The authors in [22] propose an off-policy actor-critic RL algorithm based on a maximum entropy reinforcement learning framework. The participants' goal in this framework is to maximize the expected reward while maximizing the entropy. They achieved state-of-the-art sample efficiency results by combining a maximum entropy framework. However, in practice, the commonly used off-policy approximate dynamic programming methods based on the Q-learning and actor-critic methods are susceptible to data distribution. They can only make limited progress without collecting additional on-policy data. To address this problem, the authors in [23] proposed bootstrap error accumulation reduction to reduce off-policy algorithm instability caused by accumulating backup operators via the Bellman algorithm. The authors in [24] developed a new estimator called offline dual reinforcement learning, which is based on the cross-folding estimation of Q-functions and marginalized density ratios. The authors in [25] used a framework combining imitation learning and deep reinforcement learning, effectively reducing the RL algorithm's instability. The authors in [26] used the DQN replay datasets to study off-policy RL, effectively reducing the off-policy algorithm's instability. The authors in [27] proposed an intelligent routing algorithm combining the graph neural network (GNN) and deep deterministic policy gradient (DDPG) in the SDN environment, which can be effectively extended to different network topologies, improving load-balancing capabilities and generalizability. The authors in [28] combined GNN with the DQN algorithm to address the lack of generalization abilities in untrained OTN topologies. OTN topology graphs are non-Euclidean data, and the nodes in their topology graphs typically contain useful feature information that most neural networks are unable to comprehend. They use MPNN to extract feature information between OTN topological nodes, which improves the generalization performance of the DRL algorithm.

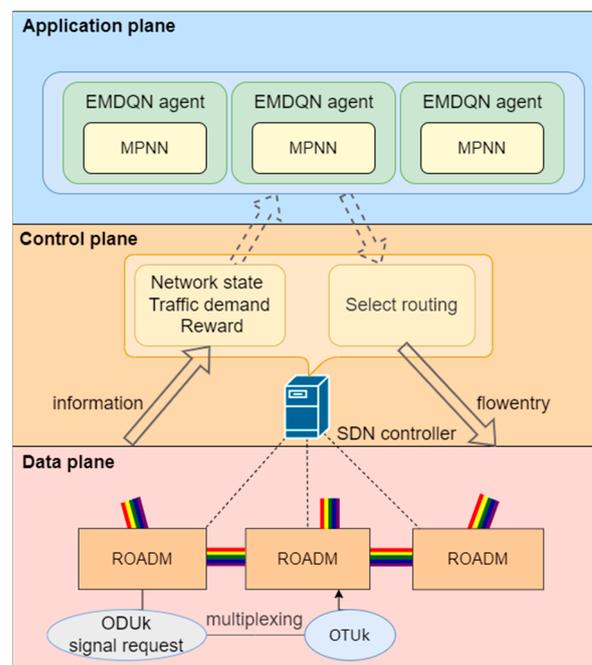
However, it is a challenge for a single DRL agent to balance exploration and development, resulting in limited convergence performance. Ensemble learning solves a single prediction problem by building several models. It works by generating several classifiers or models, each of which learns and predicts independently. These predictions are finally combined into a combined prediction, which outperforms any single classification for making predictions [29]. There are two types of integrated base learning machines. One type involves using various learning algorithms on the same dataset to obtain a base learning machine, which is usually referred to as heterogeneous [30–32]. The other type applies the same learning algorithm on a different training set (which can be obtained by random sampling based on the original training dataset, etc.), and the base learning machine obtained using this method is said to be a homogeneous type. However, because of the high implementation difficulty and low scalability of heterogeneous types of base learning machines, expansion to high-dimensional state and action spaces is difficult, making it unsuitable for solving OTN routing optimization problems. Table 1 summarizes the description of the papers reviewed, whether SDN and RL are considered, and the evaluation indicators. The EMDQN algorithm we propose applies the same reinforcement learning algorithm to different training sets to generate the base learning machine. We combine multiple EMDQN agents to construct an ensemble learning machine and generate diverse samples to effectively generate learning machines with high generalization abilities and significant differences.

**Table 1.** Related work.

Paper	Description	RL	DL	OTN	Evaluating Indicator
[7]	Performance analysis of OSPF				Network convergence, traffic dropped
[8]	Embarks upon a systematic algorithmic study of traffic engineering with ECMP				Throughput
[9]	Allocation of computational resources based on heuristic ant colony algorithm				Latency, load balancing, task completion time.
[10]	A load-balancing algorithm based on the alienated ant algorithm				Throughput, delay, packet loss rate
[11]	SDN routing solution about flow feature extraction, requirement prediction and route selection				Routing efficiency
[12]	An OTN network planning solution over DWDM based on heuristic algorithms			✓	Network resource consumption
[13]	A heuristic algorithm of minimum cost tree for path sharing and load balancing			✓	Tree cost, run time, degree of load balancing
[14]	A network traffic prediction model based on LSTM		✓		Throughput, load-balancing degree
[16]	Deep learning classifier for detection of anomalies		✓		Precision, recall, accuracy of classification
[17]	A Q-learning-based localization-free anypath routing	✓			Delay, network lifetime, packet delivery ratio
[19]	Combines the control theory and DRL technology to achieve an efficient network control scheme	✓	✓		Transmission delay
[20]	An RL routing algorithm to solve a traffic engineering	✓	✓		Throughput and delay, transmission time
[21]	Designing state and action to simplify the DRL algorithm	✓	✓	✓	Link utilization
[27]	A set of extensions to the MQTT protocol that meet application-defined real-time requirements	✓	✓		Latency
[28]	A DRL algorithm combined with GNN	✓	✓	✓	Network capacity
[30]	A new method of data missing estimation with tensor heterogeneous ensemble learning		✓		Data missing rates
[32]	A method to automatically learn long-term associations between traffic samples		✓		Calculates precision, recall and F1-score

### 3. SD-OTN Architecture

In this paper, the designed SD-OTN architecture consists of the application, control, and data planes, as shown in Figure 1. The description of each part of the network architecture is as follows:



**Figure 1.** The SD-OTN architecture. The architecture consists of the application plane, control plane, and data plane.

1. **Data plane.** The data plane consists of the ROADM nodes and the predefined optical paths connecting them. In the data plane, the capacity of the links and the connection status of the ROADM nodes are predefined. The data plane must collect the current optical data unit (ODU) signal requests and network status information, which it must then send to the control plane via the southbound interface. The data plane implements the new routing forwarding policy after receiving it from the control plane. It communicates the new network state and traffic demand to the control plane, from which decision-makers in the application plane learn.
2. **Control plane.** The control plane consists of the SDN controller. The control plane obtains the ODU signal request and network status information via the southbound interface and calculates the reward using the reward function. Through the northbound interface, the control plane sends the network state, traffic demand, and reward to the application plane via the northbound interface. When receiving optimized routing action from the application plane, the control plane sends a routing forwarding policy to the data plane based on the routing action.
3. **Application plane.** The application plane manages the EMDQN agents. The agents obtain network state information from the control plane, encode it, and feed it into the agents' policy network, which generates optimized routing actions. Subsequently, the routing actions are sent down to the control plane.

#### 4. EMDQN-Based Decision Design for Routing Optimization

In this section, we describe in detail the EMDQN algorithm proposed in this paper.

##### 4.1. DRL-Based Routing Optimization in SD-OTN

Based on the system architecture described above, the DRL agent's role is to assign routes to incoming traffic demands for a specific sequence of optical paths (i.e., end-to-end paths) to maximize network utility. Because the DRL agent operates in the electrical domain, traffic demands are treated as requests for ODU signals. These signals, which may originate from different clients, are multiplexed into an optical transform unit (OTU), as shown in Figure 1. The final OTU frames are transmitted through the optical channels in the OTN [33].

We use  $G$  to refer to an optical transmission network, as shown in Equation (1):

$$G = (V, E) \quad (1)$$

where  $V$  and  $E$  represent the set of  $n$  ROADM nodes and  $m$  optical links in the network topology, respectively, as shown in Equations (2) and (3).

$$V = [v_1, v_2, \dots, v_n]. \quad (2)$$

$$E = [e_1, e_2, \dots, e_m]. \quad (3)$$

We use  $C$  to denote the set of link bandwidth capacity, as shown in Equation (4), where  $|C| = |E| = m$ :

$$C = [c_1, c_2, \dots, c_m]. \quad (4)$$

The path  $k$  from node  $v_i$  to node  $v_j$  is defined as a sequence of links, as shown in Equation (5), where  $e_{k(i)} \in E$ :

$$p_k = \{e_{k(0)}, e_{k(1)}, \dots, e_{k(n)}\}. \quad (5)$$

We use  $d_k$  to denote the traffic demand of the path  $k$ , and define  $D$  as the set of all traffic demands, as shown in Equation (6):

$$D = [d_1, d_2, \dots, d_{n*n}]. \quad (6)$$

The traffic routing problem in OTN is a classical resource allocation problem [26]. If the bandwidth capacity of the distributed routing path is greater than the size of the bandwidth requirement, the allocation is successful. After successfully allocating bandwidth capacity for a node pair's traffic demand, the routing path will not be able to release the bandwidth occupied by that demand until the end of this episode. We use  $rb_i$  to describe the remaining bandwidth of the link  $e_i$ , which is the link bandwidth capacity  $c_i$  minus the traffic demands of all paths passing through link  $e_i$ , as shown in Equation (7).  $RB$  is the set of the remaining bandwidth of all links, as shown in Equation (8).

$$rb_i = c_i - \sum d_k. \quad (7)$$

$$RB = [rb_1, rb_2, \dots, rb_m]. \quad (8)$$

We use  $q_k$  to denote the allocating traffic demand of the path  $k$ , as shown in Equation (9).  $Q$  is the set of all allocating traffic demands, as shown in Equation (10).

$$q_k = \begin{cases} d_k, & \text{if } \forall e \in p_k \text{ and } r_e > d_k \\ 0, & \text{else} \end{cases}. \quad (9)$$

$$Q = [q_1, q_2, \dots, q_{n*n}]. \quad (10)$$

The optimization objective in this paper is to successfully allocate as much of the traffic demand as possible, as shown in Equation (11):

$$\max(\sum_{q_i \in Q} q_i). \quad (11)$$

In view of the above optimization objective, the routing optimization can be modeled as a Markov decision process, defined by the tuple  $\{S, A, P, R\}$ , where  $S$  is the state space,  $A$  is the action space,  $P$  is the set of transfer probabilities, and  $R$  is the set of rewards. The specific design is as follows:

1. Action space: The action space is designed as  $k$  shortest hop-based paths of source-destination nodes. The action selects one of the  $k$  paths to transmit the traffic demand

of source–destination nodes. The parameter  $k$  is customizable and varies according to the topology’s complexity. The action space is invariant to the arrangement of nodes and edges, which is discretely distributed, allowing the DRL agent to understand the actions on arbitrary network states easily.

2. State space: The state space is designed as the remaining bandwidth  $RB$ , the traffic demand  $D$ , and the link betweenness. The link betweenness is a centrality metric, which indicates how many paths are likely to cross the link. For each node pair in the topology,  $k$  candidate shortest routes are calculated, with the link betweenness value being the number of shortest routes passing through the link divided by the total number of paths, as shown in Equation (12), where  $bn_i$  represents the betweenness of the link  $e_i$ ,  $N$  represents the total number of paths,  $p_i^k$  represents the number of shortest routes passing through the link  $e_i$  in  $k$  candidate shortest routes:

$$bn_i = p_i^k / N. \quad (12)$$

3. Reward function: The reward function returns a positive reward if the selected link has sufficient capacity to support the traffic demand in an episode; otherwise, it returns no reward and terminates the episode. According to the optimization objective in Equation (11), the final reward for the episode is the sum of the rewards of all successfully assigned traffic demand tuples  $\{src, dst, demand\}$ , as shown in Equation (13), where  $N$  is the number of traffic demand tuples,  $r_t$  represents the reward after the action at time  $t$ ,  $q_i$  represents the  $i$ -th traffic demand successfully assigned, and  $q_{max}$  represents the maximum traffic demand successfully assigned. The higher the reward, the more bandwidth demands are successfully allocated in that time step, and the better the network load-balancing capability.

$$r_t = \sum_{i=1}^N q_i / q_{max}. \quad (13)$$

#### 4.2. DQN Algorithm

Based on the above DRL-based optimization solution, this paper selects the DQN algorithm to implement a reinforcement learning agent. The DQN is a classical DRL algorithm based on value functions. It combines a convolutional neural network (CNN) with the Q-learning algorithm, using the CNN model to output the Q-value corresponding to each action to ascertain which to perform [6].

The DQN algorithm uses two network models containing CNNs for learning: the prediction network  $Q(s, a, \theta)$  and the target network  $\hat{Q}(s, a, \bar{\theta})$ , where  $\theta$  and  $\bar{\theta}$  are the network parameters of the prediction and target networks, respectively. The prediction network outputs the predicted Q-value corresponding to the action, whereas the target network calculates the target value and updates the parameters of the prediction network based on a loss function. The DQN copies the parameters of the prediction network model to the target network after each  $C$ -round iteration.

The prediction network approximates the action value function through the CNN model  $Q_\pi(s, a)$ , as shown in Equation (14):

$$Q(s, a, \theta) \approx Q_\pi(s, a). \quad (14)$$

The DQN agent selects and executes an action based on an  $\epsilon$ -greedy policy. The policy generates a random number in  $[0, 1]$  interval through a uniform distribution. If the number

is less than  $1 - \epsilon$ , it selects an action that maximizes the Q-value; otherwise, it selects an action randomly, as shown in Equation (15):

$$a_t = \begin{cases} \operatorname{argmax}_a Q(s_t, a, \theta), & \text{with probability } 1 - \epsilon \\ \text{random action,} & \text{otherwise} \end{cases}. \quad (15)$$

The target network calculates the target value  $y$  by obtaining a random mini-batch storage sample from the replay buffer, as shown in Equation (16), where  $r$  is the reward value and  $\gamma$  is the discount factor:

$$y = r + \gamma \max_{a'} \hat{Q}(s', a', \bar{\theta}). \quad (16)$$

The DQN defines the loss function of the network using the mean-square error, as shown in Equation (17). The parameter  $\theta$  is updated by the mini-batch semi-gradient descent, as shown in Equations (18) and (19):

$$L(\theta) = \mathbf{E}[(y - Q(s, a, \theta))^2], \quad (17)$$

$$\nabla_{\theta} L(\theta) \approx \frac{1}{N} \sum_i^N (y - Q(s, a, \theta)) \nabla Q(s, a, \theta), \quad (18)$$

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} L(\theta), \quad (19)$$

where  $N$  represents the number of samples and  $\alpha$  represents the update parameter.

The target network is used by the DQN to keep the target Q-value constant over time, which reduces the correlation between the predicted and target Q-values to a certain extent. This operation reduces the possibility of loss value oscillation and divergence during training and improves the algorithm's stability.

#### 4.3. Message-Passing Neural Network

The CNN model has better results in extracting features from Euclidean spatial data (e.g., picture data), characterized by a stable structure and dimensionality. However, graph-structured or topologically structured data are infinitely dimensional and irregular, and the network surrounding each node may be unique. Such structured data renders traditional CNNs ineffective and unable to extract data features effectively. To address this problem, we use the MPNN rather than the CNN as a network model for the DQN. The MPNN is a type of GNN that is suitable for extracting spatial features of topological graph data [5].

Through repeated iterations of the process of passing data about the link's hidden state, the MPNN algorithm abstracts information about the characteristics of the network. The characteristic values of the hidden state  $h_i$  include the remaining bandwidth  $rb_i$ , the link betweenness  $bn_i$ , and the traffic demand feature  $df_i$ . The traffic demand feature  $df_i$  represents the quantitative characteristics of the traffic demand  $d_i$ . Because the traffic demand of the OTN environment is discrete and finite, the traffic demand feature is denoted by an  $n$ -element one-hot encoding, and link characteristics that are not included in the  $k$  routes have a zero value. Additionally, the size of the hidden state is usually larger than the size of the feature values in the hidden state; thus, we use zero values to populate the feature value vector, as shown in Equation (20):

$$h_i = [rb_i, bn_i, df_i, 0, \dots, 0]. \quad (20)$$

The MPNN workflow is shown in Figure 2. We perform a message-passing process between all links which will be executed  $T$  times. First, the MPNN receives link hidden features as the input. Second, each link iterates over all of its adjacent links to obtain the link features. In the message-passing process, for each link  $k$ , we generate messages by entering the hidden state  $h_k$  of the link and the hidden state  $h_i$  of all neighboring links into

the message function  $m(\cdot)$ . The message function  $m(\cdot)$  is a fully connected CNN. After iterating over all links, the link  $k$  receives messages from all neighboring links (denoted by  $N(k)$ ). It generates a new feature vector  $M_k$  using message aggregation, as shown in Equation (21):

$$M_k^{t+1} = \sum_{i \in N(k)} m(h_k^t, h_i^t), \quad (21)$$

where  $N(k)$  represents all neighboring links of the link  $k$ .

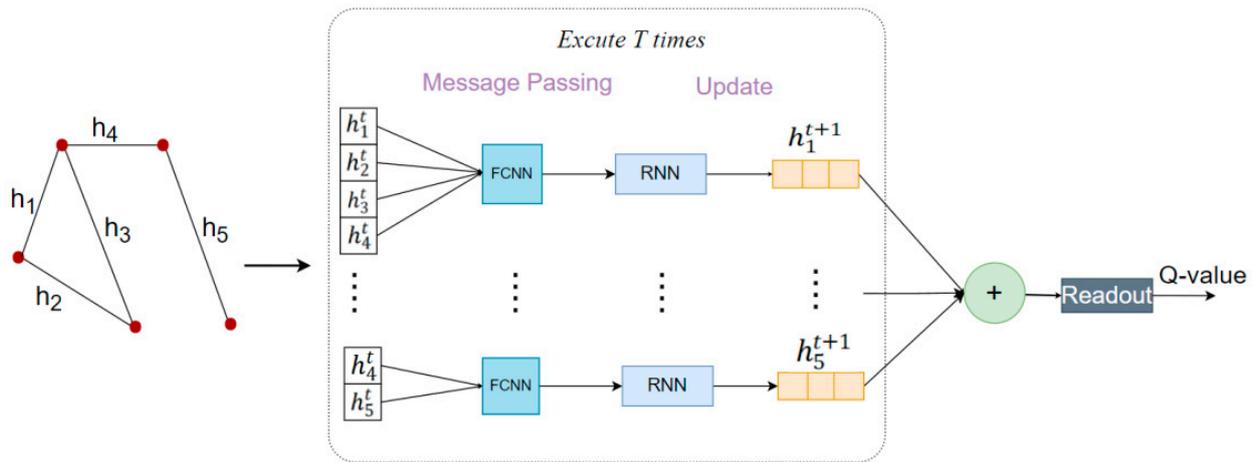


Figure 2. The MPNN workflow.

Second, we update the hidden state of the link by aggregating the feature vector  $M_k^{t+1}$  with the link-hidden state  $h_k^t$  through the update function  $u(\cdot)$ , as shown in Equation (22). The update function  $u(\cdot)$  is the Gate Recurrent Unit (GRU), which has the characteristics of high training efficiency.

$$h_k^{t+1} = u(h_k^t, M_k^{t+1}). \quad (22)$$

Finally, after the T-step message transmission, we use the readout function  $R(\cdot)$  to aggregate the hidden state of all links and obtain the Q-value, as shown in Equation (23):

$$Q(s, a, \theta) = R\left(\sum_{k \in E} h_k\right), \quad (23)$$

where  $E$  represents the set of all links in the topology.

#### 4.4. Ensemble Learning

In the DQN algorithm, it is challenging for a single agent to balance exploration and development, resulting in limited convergence performance. Furthermore, errors in the DQN target values can increase the overall error in the Q-function, leading to an unstable convergence. In this paper, we use ensemble learning to solve the above problems. Ensemble learning has the advantage of efficient exploration and can reduce uncertainty in new samples.

As shown in Figure 3, ensemble learning is realized by a set of multiple EMDQN agents  $\{Q(s, a, \theta_i)\}_{i=1}^N$ , where  $\theta_i$  represents the parameter of the  $i$ -th agent. To diversify the training of the EMDQN agents, we randomly initialize the policy network of all EMDQN agents. In the training phase, we employ the  $\epsilon$ -greedy-based upper-confidence bound (UCB) exploration strategy [34], as shown in Equation (24):

$$a_t = \begin{cases} \max_a \{Q_{\text{mean}}(s_t, a, \theta) + \lambda Q_{\text{std}}(s_t, a, \theta)\}, & \text{with probability } 1 - \epsilon \\ \text{random action}, & \text{otherwise} \end{cases}, \quad (24)$$

where  $Q_{\text{mean}}(s_t, a, \theta)$  and  $Q_{\text{std}}(s_t, a, \theta)$  are the mean and standard deviation of the Q-values output by all MPNN policy networks  $\{Q(s, a, \theta_i)\}_{i=1}^N$ . The exploration reward  $\lambda > 0$  is a hyper-parameter. When  $\lambda$  increases, the EMDQN agents become more active in accessing unknown state–action pairs.

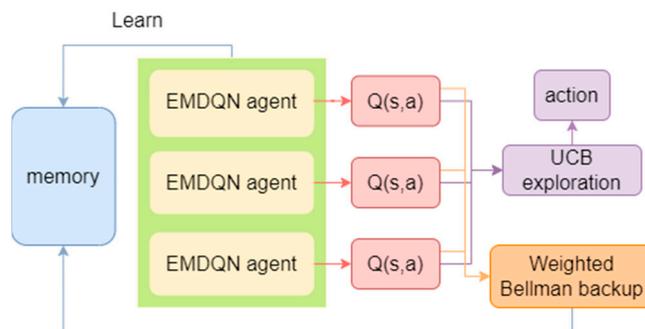


Figure 3. EMDQN workflow.

The traditional DQN loss function (Equation (6)) may be affected by error propagation, that is, it propagates the target Q-network  $\hat{Q}(s', a', \bar{\theta})$  error to the current state of the Q-network  $Q(s, a, \theta)$ . This error propagation can lead to an unstable convergence. To alleviate this problem, for each EMDQN agent  $i$ , this paper uses Bellman weighted backups, as shown in Equation (25):

$$L_{WQ}^{\text{EMDQN}}(\theta_i) = w(s) \left( r + \gamma \max_{a'} \hat{Q}(s', a', \bar{\theta}_i) - Q(s, a, \theta_i) \right)^2, \tag{25}$$

where  $w(s)$  represents the confidence weight of the set of target Q-networks in the interval  $[0.5, 1.0]$ .  $w(s)$  is calculated from Equation (26), where the weight parameter  $W$  is a hyper-parameter,  $\sigma$  is a sigmoid function,  $\hat{Q}_{\text{std}}(s)$  is the empirical standard deviation of all target Q-networks  $\left\{ \max_a \hat{Q}(s, a, \bar{\theta}) \right\}_{i=1}^N$ .  $L_{WQ}^{\text{EMDQN}}(\cdot)$  reduces the weights of sample transitions with high variance between target Q-networks, resulting in better signal-to-noise ratios for network updates.

$$w(s) = \sigma(-\hat{Q}_{\text{std}}(s) * W) + 0.5. \tag{26}$$

#### 4.5. The Working Process of the EMDQN Agent

The working process of the EMDQN agent at each iteration is described in Algorithm 1. We first reset the environment and obtain the environment link capacity and traffic demand tuple  $\{src, dst, demand\}$  (line 1). Subsequently, we execute a loop to continuously assign traffic demands. In the process, we compute  $k$  shortest links (Line 3) and allocate the traffic demand for each shortest link through  $k$  cycles (Lines 4–8). Based on this, we can compute the Q-value for each action. We select actions using an  $\epsilon$ -greedy-based UCB exploration strategy (Line 9); subsequently, we apply the chosen route to the environment (Line 10). We store the rewards and state transfer during the interaction with the environment in the experience replay buffer (Line 11) while applying the transferred state (Line 12). The cycle stops when any link is unable to carry the traffic demand. Next, we execute the agent learning phase. For the sampled batch (Line 15), we plot the mask using the Bernoulli distribution (Line 16) and calculate the batch weight using all EMDQN agents. Following that, we multiply the sample by the weight and mask to minimize  $L_{WQ}^{\text{EMDQN}}$  (Line 18). Finally, we evaluate the set of EMDQN agents in the environment (Line 21) and collect the rewards, as well as the status of the environment, in the evaluation process to analyze the training situation of the EMDQN agent.

**Algorithm 1:** working process of the EMDQN agent

---

```

1:  $s, demand, src, dst \leftarrow env.reset()$ 
2: while (Done  $\neq$  False) do
3:    $k\_path \leftarrow compute\_k\_path(k, src, dst)$ 
4:   for  $i \leftarrow 1$  to  $k$  do
5:      $path \leftarrow get\_path(i, k\_path)$ 
6:      $s' \leftarrow allocate(s, path, src, dst, demand)$ 
7:      $k\_Q[i] \leftarrow compute\_Q(s', path)$ 
8:   end for
9:    $a \leftarrow act(k\_Q, \epsilon, k\_path, s)$ 
10:   $s', r, done, demand', src', dst' \leftarrow env.step(s, a)$ 
11:   $agent.remember(s, a, r, s', done)$ 
12:   $s, demand, src, dst \leftarrow s', demand', src', dst'$ 
13: end while
14: for  $i \leftarrow 1$  to STEP do
15:   $batch \leftarrow sample()$ 
16:   $m \leftarrow bernoulli()$ 
17:  for each agent  $i$  do
18:    Update agent by minimizing  $L_{EMDQN}^{WQ}(\theta_i)$ 
19:  end for
20: end for
21:  $agent.evaluate()$ 

```

---

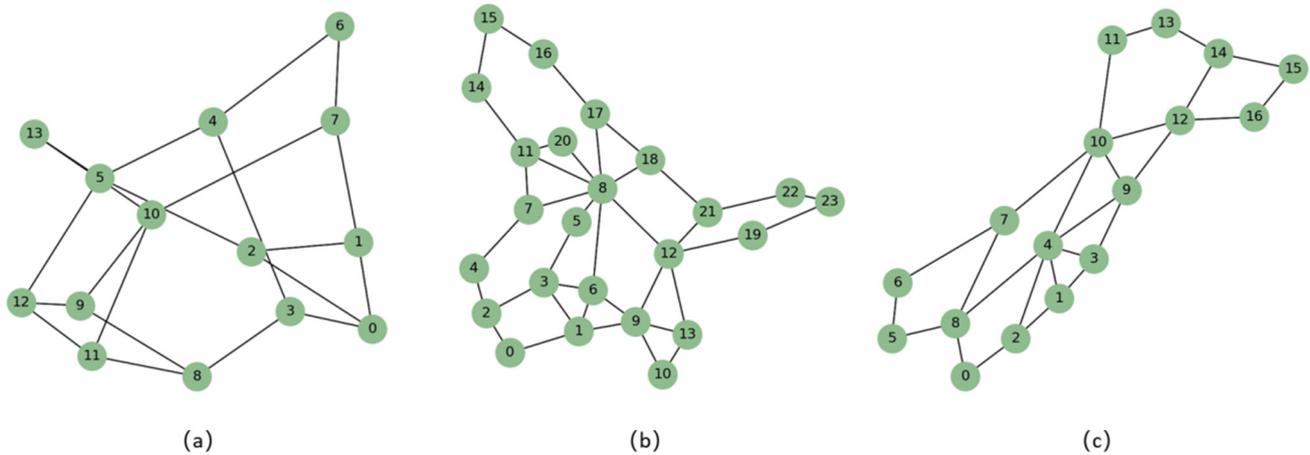
**5. Experiments and Analysis**

In this section, we simulate the SD-OTN routing scenario using the OpenAI gym framework to train and evaluate the EMDQN algorithm. Furthermore, we conduct experiments and analyses by adjusting the hyper-parameters and evaluating the algorithm load-balancing ability and generalization ability.

*5.1. Experimental Environment*

The computer used for the experiments has an AMD R5 5600G processor with a base frequency of 2900 MHz, a 2 TB solid-state drive, and 32 GB of RAM. The experiment uses the Tensorflow deep learning framework to implement the EMDQN algorithm. We select NSFNET, GEANT2, and GBN for the optical transmission network topology, with the lightpath bandwidth being 200 ODU0 bandwidth units, as shown in Figure 4. Among these, the NSFNET network contains 14 ROADM nodes and 21 lightpaths, the GEANT2 network contains 24 ROADM nodes and 36 lightpaths, and the GBN network contains 17 ROADM nodes and 27 lightpaths.

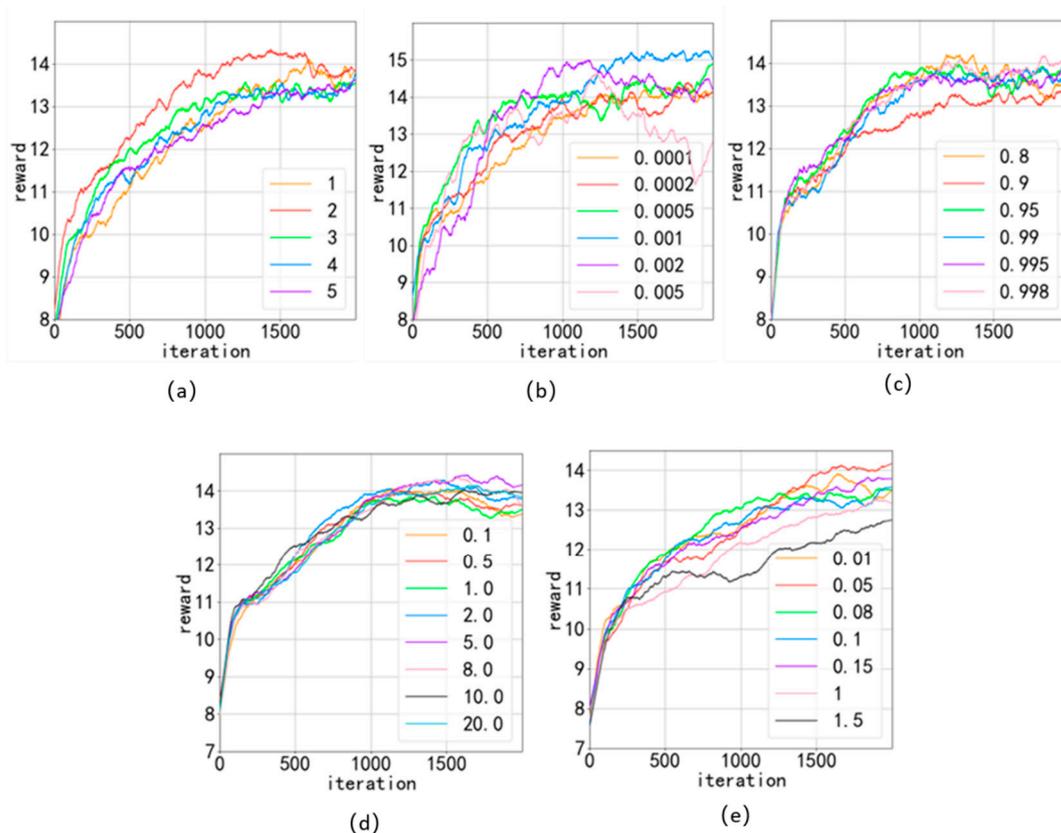
In this paper, the lightpath bandwidth requirements are expressed as multiples of ODU0 signals, i.e., 8, 32, and 64 ODU0 bandwidth units. In each episode, the environment generates a traffic demand tuple  $\{src, dst, demand\}$  at random. Additionally, the EMDQN agent should assign the appropriate route for each tuple received. If the assignment is successful, it will receive a reward as defined in Equation (1). Otherwise, it will not be rewarded. Since the new traffic demand is randomly generated, the routing policy designed by the EMDQN agent does not rely on traffic demand distribution information, reducing the EMDQN agent's overfitting to the particular network scenario used for training.



**Figure 4.** The optical transmission network topologies: (a) the NSFNET topology; (b) the GEANT2 topology; (c) the GBN topology.

### 5.2. Hyper-Parameters Settings

We experimentally select suitable hyper-parameters for the EMDQN agent, as shown in Figure 5. In the experiments, we chose the NSFNET as the experimental network topology. The size of the link-hidden state is related to the amount of coding information. We set the size of the link-hidden state to twenty and the number of feature values to five, and filled the rest with zero. To facilitate observation, we smoothed the data when drawing the graph.



**Figure 5.** Comparison of the effect of some super-references: (a) ensemble number; (b) learning rate; (c)  $\epsilon$ -decay; (d) UCB exploration reward  $\lambda$ ; (e) weight parameter  $W$ .

Figure 5a shows the training results for the different numbers of EMDQN agents. When the number of agents is high, the training slows down and aggravates the overfitting of DRL agents in the application scenario, resulting in poorer results. The performance is optimal when the number of EMDQN agents is two. Figure 5b shows the training results of the stochastic gradient descent algorithm with different learning rates. When the learning rate was 0.001, the algorithm reward achieved the highest value. Figure 5c shows the training results for different decay rates of  $\epsilon$ . In the initial stage of training,  $\epsilon$  is close to 1. We executed 70 iterations and started to reduce  $\epsilon$  exponentially using  $\epsilon$ -decay until it decreased to 0.05. During the process of  $\epsilon$  reduction, the training curve tends to flatten out, finally reaching convergence. The training results show that the reward value curve is most stable after convergence when  $\epsilon$ -decay is 0.995. Figure 5d shows the training results for different  $\lambda$  values in Equation (11).  $\lambda$  denotes the exploration reward of the EMDQN agent. From the results in Figure 5d, it is clear that the algorithm reward value is highest when  $\lambda$  value is 5. Figure 5e depicts the training results for different weight parameters  $W$  in Equation (14). In this paper, we set the size of samples to 32. As  $W$  increases, the sample weights converge and become less than one, which affects the sample efficiency of the EMDQN. The reward of this algorithm reaches its highest value when the value of  $W$  is 0.05.

Table 2 shows some relevant parameters of the EMDQN and values taken after tuning the parameters.

**Table 2.** Some relevant parameters of the EMDQN.

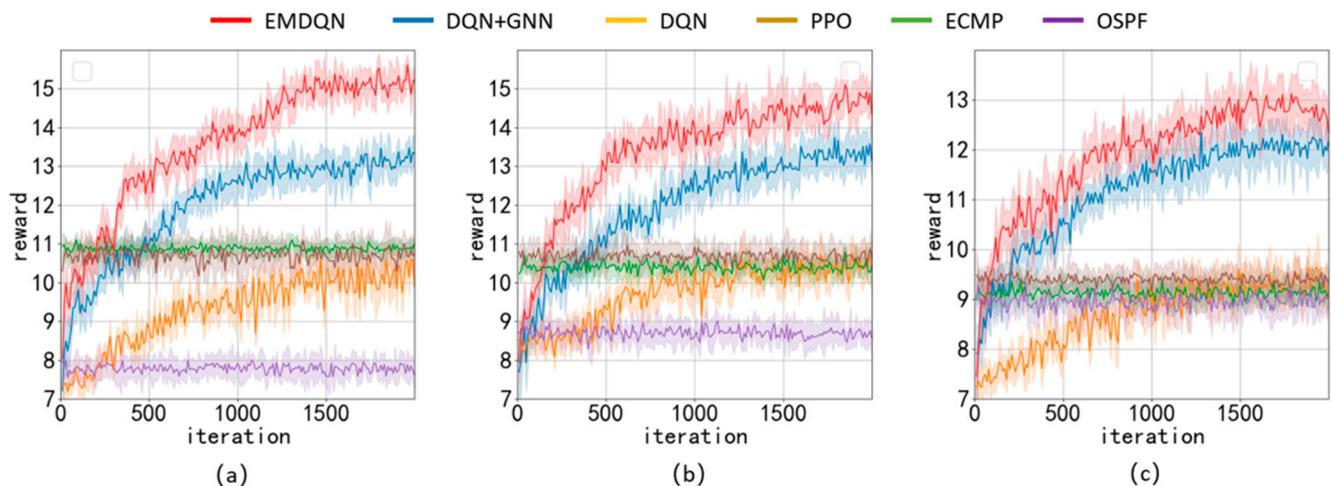
Parameter	Value
Batch size	32
Learning rate	0.001
Soft wights copy $\alpha$	0.08
Dropout rate	0.01
State hidden	20
Ensemble number	2
UCB exploration reward $\lambda$	5
Weight parameter $W$	0.05
$\epsilon$ -decay	0.995
Discount factor $\gamma$	0.95

### 5.3. Load-Balancing Performance Evaluation

In this section, we experimentally evaluate the EMDQN in the three network topologies, as described in Section 5.1. The DRL agent runs 2000 iterations. In each iteration, the agent trains 50 episodes and evaluates 40 episodes. Furthermore, the DRL agent updates the network during the training period. During the evaluation, the DRL agent does not update the network; rather, it applies the action to the environment intending to maximize the Q-function, and subsequently records network state data, such as rewards, link utilization, and throughput for each episode.

We implement other SDN solutions for performance comparison with EMDQN algorithms, such as OSPF [7], ECMP [8], DQN [17], PPO [19], and DQN+GNN [26]. The DQN+GNN is an ablation experiment among the compared algorithms, i.e., a performance comparison of the EMDQN model with ensemble learning removed. The DQN and PPO are classic DRL algorithms that use a fully connected feedforward NN as a policy network. The OSPF is an open shortest path algorithm that performs an action selection by calculating the shortest number of hops of the link traversed between the source and destination nodes. The ECMP algorithm is an equal-value multipath routing protocol that allows the use of multiple links simultaneously in the network. The ECMP algorithm distributes the bandwidth demand equally over  $k$  lightpaths in this experiment. Furthermore, OUD0 signals are not divisible, but we can verify the performance in other network scenarios in this way.

Figure 6 shows the average reward of all algorithms for the three evaluation scenarios, where the confidence interval is 95%. In this paper, we design the reward based on whether the bandwidth demand can be successfully allocated. The greater the reward, the more bandwidth demand is successfully allocated, and the better the network load-balancing capability. In all three evaluation scenarios, the EMDQN algorithm proposed in this paper performs better than other algorithms after convergence. The EMDQN algorithm outperforms the DQN+GNN algorithm with ensemble learning removed after convergence by more than 7%, demonstrating that the multi-agent ensemble learning approach can effectively improve the convergence performance of the DQN. Additionally, the EMDQN and DQN+GNN outperform the classical reinforcement learning algorithms (DQN and PPO) by more than 25% in all three evaluated scenarios. This indicates that the MPNN can effectively improve the decision performance of the reinforcement learning model by capturing information about the relationship between the demand on links and network topology. The DQN and PPO algorithms perform about as well as the ECMP algorithm after convergence. The OSPF algorithm, on the other hand, routes all flow requests singularly to the shortest path. Since this method is based on fixed forwarding rules, it can easily lead to link congestion. Therefore, the OSPF algorithm is only close to ECMP in the GBN scenario and the lowest in other scenarios.



**Figure 6.** Comparison of the rewards of each algorithm in different scenarios: (a) NSFNET scenario evaluation; (b) GEANT2 scenario evaluation; (c) GBN scenario evaluation.

Table 3 shows the average throughput of each algorithm in ODU0 bandwidth units for the three network topologies. Table 4 displays the average link utilization of each algorithm across the three network topologies. The average throughput and link utilization of the EMDQN are higher than those of other algorithms under various network topologies, indicating that the EMDQN algorithm has a better load-balancing capability for the network after convergence. The performance of the EMDQN algorithm is higher than that of the DQN+GNN algorithm, which is a good indication that ensemble learning can improve the convergence performance of the model. The results show that the EMDQN has excellent decision-making abilities.

**Table 3.** A comparison of the average throughput of each algorithm in different scenarios.

	EMDQN	DQN+GNN	DQN	PPO	ECMP	OSPF
NSFNET	1028.17 ± 27.45	899.28 ± 24.21	709.47 ± 35.47	737.48 ± 26.92	747.68 ± 13.50	548.86 ± 20.97
GEANT2	995.56 ± 35.26	903.48 ± 35.36	721.97 ± 31.07	726.89 ± 24.09	717.35 ± 19.59	605.27 ± 21.53
GBN	864.77 ± 30.28	826.06 ± 30.19	646.56 ± 29.71	652.05 ± 15.51	636.10 ± 18.69	627.70 ± 23.02

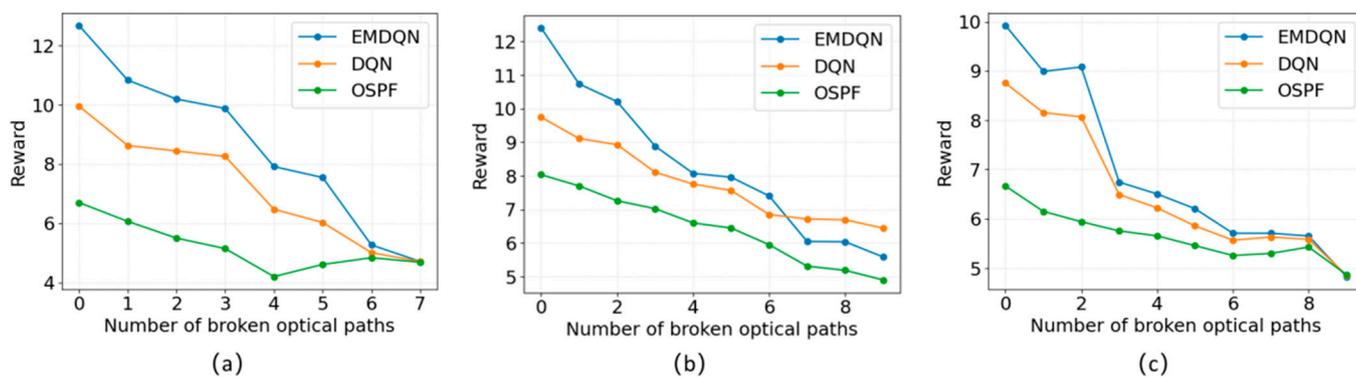
**Table 4.** A comparison of the average link utilization of each algorithm in different scenarios.

	EMDQN	DQN+GNN	DQN	PPO	ECMP	OSPF
NSFNET	56 ± 0.90%	50 ± 1.28%	40 ± 1.69%	43 ± 0.65%	41 ± 1.04%	15 ± 1.16%
GEANT2	40 ± 0.92%	36 ± 1.46%	29 ± 1.49%	30 ± 0.60%	21 ± 0.90%	11 ± 0.83%
GBN	45 ± 0.92%	42 ± 1.54%	34 ± 1.48%	35 ± 0.76%	25 ± 1.22%	15 ± 1.07%

#### 5.4. Generalization Performance Evaluation

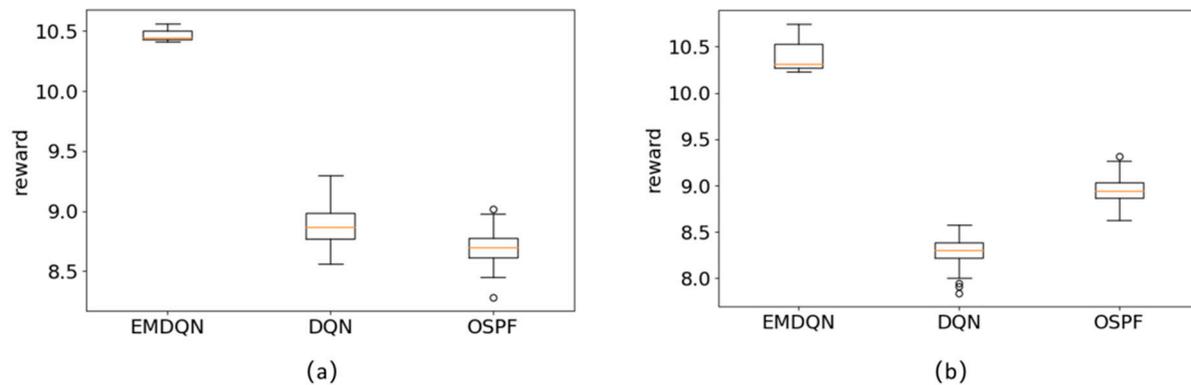
In a real OTN scenario, there is the possibility that the network topology changes due to a broken lightpath. In this case, the DRL model usually needs to be retrained, resulting in a network state that is low-load-balanced for an extended period, which is intolerable for real network situations. To verify the generalization performance of the EMDQN in this paper, we simulated the light path breakage case in the training environment. We randomly break 0–10 lightpaths in each network scenario and evaluate 100 iterations using the converged EMDQN model while ensuring that the network topology remains connected.

In the generalization experiments, we compared and analyzed the EMDQN, DQN, and OSPF. The classical DQN algorithm is implemented using a fully connected network and will fail if the network topology changes. To avoid retraining the DQN model, we removed some network parameters and applied the same evaluation method after adjusting the state inputs. Figure 7 shows the experimental results of the model's evaluation of randomly malfunctioning lightpaths in different network scenarios. When a lightpath malfunctions, a new route needs to be found to avoid the failed lightpath. As the number of faulty lightpaths increases, fewer routes become available, resulting in a reduction in network transmission traffic and a decrease in the load capacity of the network. The OSPF and the classical DQN algorithms have progressively lower rewards as the number of faulty lightpaths increases and have worse performance than the EMDQN model. In contrast, the EMDQN algorithm can still understand the state of the network and obtain a higher reward. The results demonstrate that the MPNN can still improve the model's generalization ability in the case of network failure, which allows the EMDQN agent to maintain a good performance.

**Figure 7.** Evaluation of the model in different network scenarios with randomly broken lightpaths: (a) broken lightpath at NSFNET; (b) broken lightpath at GEANT2; (c) broken lightpath at GBN.

To further verify the generalization performance of EMDQN agents, we use the EMDQN model trained to converge in NSFNET to transfer to GEANT2 and GBN network topologies for evaluation. Because of the generalization capabilities of the MPNN, the converged EMDQN agents can directly transfer to operate in different network topologies. The experimental results are shown in Figure 8. The classical DQN algorithm does not perform effectively as the traditional routing algorithm OSPF when the network topology is changed. However, the EMDQN model in this paper still works stably, and the reward section, average value, and stability are significantly better than the DQN and OSPF

algorithms. This confirms that the EMDQN agent can still maintain excellent decision-making abilities in the case of network connectivity changes.



**Figure 8.** Performance of the algorithm after changing the network topology: (a) train in the NSFNET and evaluate in the GEANT2; (b) train in the NSFNET and evaluate in the GBN.

## 6. Conclusions and Future Work

In this paper, we proposed the EMDQN algorithm, which uses the MPNN as a policy network for the DRL to improve the DRL agent's decision-making and generalization abilities, allowing the EMDQN agent to efficiently generalize the unknown topology. We verify the convergence and generalization performance of the EMDQN algorithm by SD-OTN simulating experiments, analyzing and comparing traditional routing protocols with some SDN solutions based on other DRL algorithms. The experimental results show that the EMDQN model can generalize unknown network topologies and outperform other SDN solutions. Furthermore, integrating multiple agents using integrated learning and combining weighted Bellman backup as well as UCB exploration strategies improves the convergence performance while alleviating the DRL agents' unstable operation when converging.

However, the difficulty of adjusting parameters is one challenge faced by the DRL. As seen in Section 5.3, the EMDQN has more hyper-parameters and requires several experiments to complete the adjustment parameters. Therefore, in our future work, we will continue to improve the DRL algorithm and reduce the parameter sensitivity of the DRL method to reduce the reality gap in the DRL method's landing.

**Author Contributions:** Conceptualization, J.C. and Y.Z.; methodology, J.C.; software, W.X. and X.L.; validation, J.C., W.X., and X.L.; formal analysis, X.H.; investigation, M.W.; resources, D.H.; data curation, J.C.; writing—original draft preparation, J.C., W.X., and X.L.; writing—review and editing, J.C. and Y.Z.; visualization, X.H.; supervision, Y.Z.; project administration, Y.Z.; funding acquisition, J.C. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work is supported by the major program of Guangxi Natural Science Foundation (No.2020GXNSFDA238001) and the Middle-aged and Young Teachers' Basic Ability Promotion Project of Guangxi (No.2020KY05033).

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The authors confirm that the data supporting the findings of this study are available within the article.

**Conflicts of Interest:** The authors declare that they have no conflict of interest.

## References

1. Karakus, M.; Duresi, A. Quality of service (QoS) in software defined networking (SDN). *J. Netw. Comput. Appl.* **2017**, *80*, 200–218. [[CrossRef](#)]
2. Guo, X.; Lin, H.; Li, Z.; Peng, M. Deep-Reinforcement-Learning-Based QoS-Aware Secure Routing for SDN-IoT. *IEEE Internet Things J.* **2020**, *7*, 6242–6251. [[CrossRef](#)]
3. Sun, P.; Lan, J.; Guo, Z.; Xu, Y.; Hu, Y. Improving the Scalability of Deep Reinforcement Learning-Based Routing with Control on Partial Nodes. In Proceedings of the 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2020), Barcelona, Spain, 4–8 May 2020; pp. 3557–3561.
4. Nguyen, T.G.; Phan, T.V.; Hoang, D.T.; Nguyen, T.N.; So-In, C. Federated Deep Reinforcement Learning for Traffic Monitoring in SDN-Based IoT Networks. *IEEE Trans. Cogn. Commun. Netw.* **2021**, *7*, 1048–1065. [[CrossRef](#)]
5. Gilmer, J.; Schoenholz, S.S.; Riley, P.F.; Vinyals, O.; Dahl, G.E. Neural message passing for quantum chemistry. In Proceedings of the 34th International Conference on Machine Learning (ICML 2017), Sydney, Australia, 4–11 August 2017; Volume 70, pp. 1263–1272.
6. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing Atari with deep reinforcement learning. *arXiv* **2013**, arXiv:1312.5602.
7. Ali Khan, A.; Zafrullah, M.; Hussain, M.; Ahmad, A. Performance analysis of OSPF and hybrid networks. In Proceedings of the International Symposium on Wireless Systems and Networks (ISWSN 2017), Lahore, Pakistan, 19–22 November 2017; pp. 1–4.
8. Chiesa, M.; Kindler, G.; Schapira, M. Traffic engineering with Equal-Cost-Multipath: An algorithmic perspective. *IEEE/ACM Trans. Netw.* **2017**, *25*, 779–792. [[CrossRef](#)]
9. Li, C.; Jiang, K.; Luo, Y. Dynamic placement of multiple controllers based on SDN and allocation of computational resources based on heuristic ant colony algorithm. *Knowl. Based Syst.* **2022**, *241*, 108330. [[CrossRef](#)]
10. Di Stefano, A.; Cammarata, G.; Morana, G.; Zito, D. A4SDN—Adaptive Alienated Ant Algorithm for Software-Defined Networking. In Proceedings of the 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC 2015), Krakow, Poland, 4–6 November 2015; pp. 344–350.
11. Chen, F.; Zheng, X. Machine-learning based routing pre-plan for sdn. In *International Workshop on Multi-Disciplinary Trends in Artificial Intelligence*; Springer: Cham, Switzerland, 2015; pp. 149–159.
12. Xavier, A.; Silva, J.; Martins-Filho, J.; Bastos-Filho, C.; Chaves, D.; Almeida, R.; Araujo, D.; Martins, J. Heuristic planning algorithm for sharing restoration interfaces in OTN over DWDM networks. *Opt. Fiber Technol.* **2021**, *61*, 102426. [[CrossRef](#)]
13. Fang, C.; Feng, C.; Chen, X. A heuristic algorithm for minimum cost multicast routing in OTN network. In Proceedings of the 19th Annual Wireless and Optical Communications Conference (WOCC 2010), Shanghai, China, 14–15 May 2010; pp. 1–5.
14. Chen, J.; Wang, Y.; Huang, X.; Xie, X.; Zhang, H.; Lu, X. ALBLP: Adaptive Load-Balancing Architecture Based on Link-State Prediction in Software-Defined Networking. *Wirel. Commun. Mob. Comput.* **2022**, *2022*, 8354150. [[CrossRef](#)]
15. Yan, M.; Li, S.; Chan, C.A.; Shen, Y.; Yu, Y. Mobility Prediction Using a Weighted Markov Model Based on Mobile User Classification. *Sensors* **2021**, *21*, 1740. [[CrossRef](#)]
16. Wani, A.; Revathi, S.; Khaliq, R. SDN-based intrusion detection system for IoT using deep learning classifier (IDSIoT-SDL). *CAAI Trans. Intell. Technol.* **2021**, *6*, 281–290. [[CrossRef](#)]
17. Zhou, Y.; Cao, T.; Xiang, W. Anypath Routing Protocol Design via Q-Learning for Underwater Sensor Networks. *IEEE Internet Things J.* **2021**, *8*, 8173–8190. [[CrossRef](#)]
18. Jalil, S.Q.; Rehmani, M.; Chalup, S. DQR: Deep Q-Routing in Software Defined Networks. In Proceedings of the 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, UK, 19–24 July 2020; pp. 1–8.
19. Sun, P.; Guo, Z.; Lan, J.; Li, J.; Hu, Y.; Baker, T. ScaleDRL: A scalable deep reinforcement learning approach for traffic engineering in SDN with pinning control. *Comput. Netw.* **2021**, *190*, 107891. [[CrossRef](#)]
20. Che, X.; Kang, W.; Ouyang, Y.; Yang, K.; Li, J. SDN Routing Optimization Algorithm Based on Reinforcement Learning. *Comput. Eng. Appl.* **2021**, *57*, 93–98.
21. Suárez-Varela, J.; Mestres, A.; Yu, J.; Kuang, L.; Feng, H.; Barlet-Ros, P.; Cabellos-Aparicio, A. Routing based on deep reinforcement learning in optical transport networks. In Proceedings of the 2019 Optical Fiber Communications Conference and Exhibition (OFC), San Diego, CA, USA, 3–7 March 2019; pp. 1–3.
22. Haarnoja, T.; Zhou, A.; Abbeel, P.; Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Proceedings of the 35th International Conference on Machine Learning (ICML 2018), Stockholm, Sweden, 10–15 July 2018; Volume 80, pp. 1861–1870.
23. Kumar, A.; Fu, J.; Soh, M.; Tucker, G.; Levine, S. Stabilizing off-policy Q-learning via bootstrapping error reduction. In Proceedings of the 33rd International Conference on Neural Information Processing Systems, Vancouver, BC, Canada, 8–14 December 2019; pp. 11784–11794.
24. Kallus, N.; Uehara, M. Double reinforcement learning for efficient off-policy evaluation in Markov decision processes. *J. Mach. Learn. Res.* **2002**, *21*, 6742–6804.
25. Qiang, F.; Xin, X.; Xitong, W.; Yujun, Z. Target-driven visual navigation in indoor scenes using reinforcement learning and imitation learning. *CAAI Trans. Intell. Technol.* **2022**, *7*, 167–176.
26. Agarwal, R.; Schuurmans, D.; Norouzi, M. An optimistic perspective on offline reinforcement learning. In Proceedings of the 37th International Conference on Machine Learning (ICML 2020), Virtual Event, 13–18 July 2020; pp. 104–114.

27. Shahri, E.; Pedreiras, P.; Almeida, L. Extending MQTT with Real-Time Communication Services Based on SDN. *Sensors* **2022**, *22*, 3162. [[CrossRef](#)] [[PubMed](#)]
28. 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* **2020**, arXiv:1910.07421v2. [[CrossRef](#)]
29. Dong, X.; Yu, Z.; Cao, W.; Shi, Y.; Ma, Q. A survey on ensemble learning. *Front. Comput. Sci.* **2020**, *14*, 241–258. [[CrossRef](#)]
30. Zhang, T.; Zhang, D.; Yan, H.; Qiu, J.; Gao, J. A new method of data missing estimation with FNN-based tensor heterogeneous ensemble learning for internet of vehicle. *Neurocomputing* **2021**, *420*, 98–110. [[CrossRef](#)]
31. Fang, Z.; Wang, Y.; Peng, L.; Hong, H. A comparative study of heterogeneous ensemble-learning techniques for landslide susceptibility mapping. *Int. J. Geogr. Inf. Sci.* **2021**, *35*, 321–347. [[CrossRef](#)]
32. Lei, L.; Kou, L.; Zhan, X.; Zhang, J.; Ren, Y. An Anomaly Detection Algorithm Based on Ensemble Learning for 5G Environment. *Sensors* **2022**, *22*, 7436. [[CrossRef](#)] [[PubMed](#)]
33. Strand, J.; Chiu, A.; Tkach, R. Issues for routing in the optical layer. *IEEE Commun. Mag.* **2001**, *39*, 81–87. [[CrossRef](#)]
34. Chen, R.; Sidor, S.; Abbeel, P.; Schulman, J. UCB exploration via Q-ensembles. *arXiv* **2017**, arXiv:1706.01502.